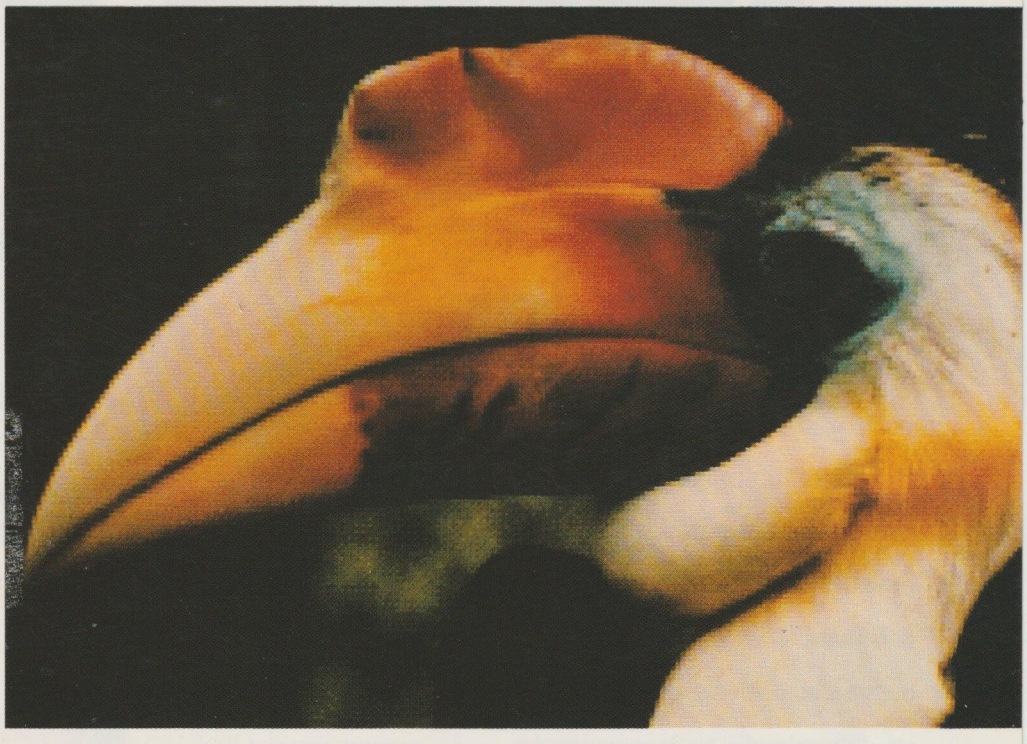
INFOPC
langages & systèmes

DÉC./JANVIER 1992 - Nº 5
SUPPLÉMENT
DÉVELOPPEURS
DU Nº 77 D'INFOPC

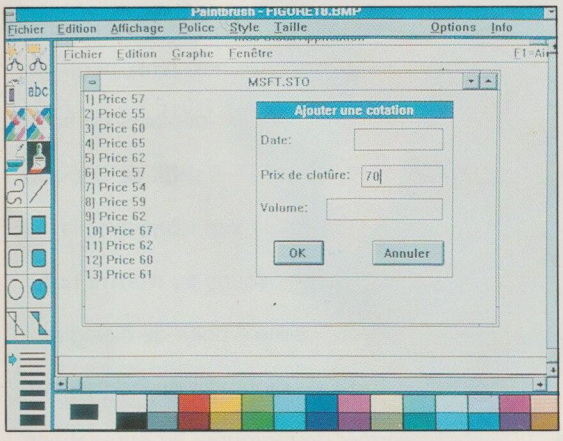
VESA

■ Le but du comité : normaliser, par le biais du Bios, la gestion logicielle des nouvelles cartes VGA



APPRENDRE WINDOWS

MSJ Un second article consacré aux boîtes de dialogue et aux contrôles de Microsoft Windows



DU C AU C++

■ Parmi les futurs langages de développement, un langage semble apte à remplacer le C, il s'agit du C++ d'AT&T

ET AUSSI...

- **LEX ET YACC, LA PRATIQUE DE LA COMPILATION**
- **LOGIN SCRIPT : L'AUTOEXEC RÉSEAU**

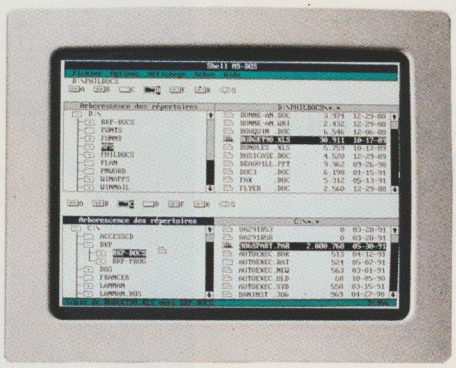
NE PEUT ÊTRE VENDU SÉPARÉMENT

Nouveau Micros votre micro rajeunir pour mo

Rajeunir ! Retrouver l'énergie et la souplesse de la jeunesse, c'est ce que va pouvoir réaliser votre plus fidèle allié : votre micro. C'est pour lui que Microsoft lance aujourd'hui la Mise à Jour MS-DOS 5.0 conçue pour les systèmes d'exploitation des micros IBM PC, PS/2 et compatibles.

Bien que la mémoire conventionnelle de votre PC soit loin d'être défaillante, lui en donner plus lui permettra d'accueillir des applications à court de mémoire ou de repousser les limites de taille de fichiers. Sur un PC 80286 ou i386™, MS-DOS 5.0 augmente ainsi la mémoire disponible d'au moins 45 Ko dans la plupart des configurations.

MS-DOS 5.0 dispose d'un centre de contrôle totalement renouvelé à partir duquel peuvent être exécutées toutes les commandes DOS. Ce "Shell" MS-DOS, avec son interface à menus déroulants et son support complet de la souris, constitue un environnement beaucoup plus convivial que la célèbre ligne de commande DOS.



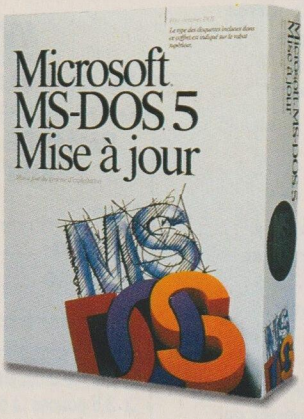
Avec MS-DOS 5.0, Microsoft s'est attaché à rendre les manipulations les plus courantes à la fois simples et performantes. Ainsi, un système complet d'aide est accessible à tout moment, soit par le menu dans le "Shell," soit en une seule instruction à la ligne de commande. Dans le Gestionnaire de Fichiers, répertoires et fichiers se présentent sous forme d'une arborescence d'icônes. Ainsi, pour déplacer un fichier, il suffit de déplacer son icône avec la souris ou d'ouvrir une simple boîte de dialogue. Le Gestionnaire de Programmes permet de regrouper les applications en fonction de votre propre organisation.

ROUX SEGUELA CAYZAC & GOUDARD

oft MS-DOS 5.0 : va se sentir ins de 790 F. ttc.*

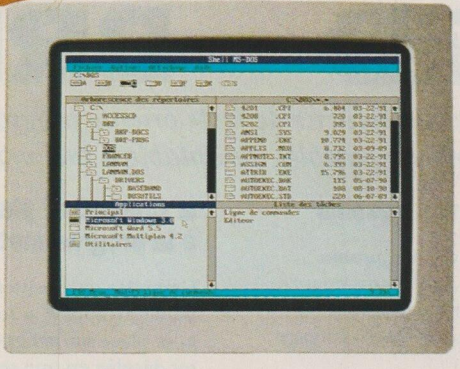
Et, avec la nouvelle fonction de Commutation de Tâches, vous pouvez charger puis mettre en attente plusieurs applications sans qu'il soit nécessaire de les fermer pour passer de l'une à l'autre.

L'ancien éditeur de ligne "Edlin" est remplacé par un nouvel éditeur pleine page à menus déroulants "Edit." Succédant à l'interpréteur GW-Basic, le MS-DOS QBasic est un langage de programmation Basic moderne qui offre rapidité et simplicité aux utilisateurs intéressés par la programmation structurée.



Pour vous garantir contre toutes les erreurs de manipulation, MS-DOS 5.0 dispose également d'utilitaires sophistiqués tels que les commandes "Undelete" (récupération de fichiers effacés), "Unformat" (annulation de formatage de disque), ou encore la fonction de reformatage rapide d'une disquette.

Enfin, la Mise à Jour MS-DOS 5.0 est si facile à installer (quelle que soit votre version antérieure de MS-DOS à partir de la 2.10) que vous pourrez remettre à niveau chaque PC de votre parc en un clin d'œil. L'installation peut même se faire à travers le réseau lors d'une session de connexion. De plus, le package de Mise à Jour dispose d'une procédure de désinstallation vous permettant de retrouver, si besoin était, votre configuration initiale équipée de votre ancienne version de DOS. Grâce à un test intensif de plusieurs mois sur plus de 7000 sites à travers le monde, la Mise à Jour MS-DOS 5.0 est le logiciel le plus testé de l'histoire de l'informatique avant sa mise sur le marché. Toutes les configurations matérielles et logicielles possibles ont été envisagées pour offrir une fiabilité maximale à votre configuration.



La Mise à Jour Microsoft MS-DOS 5.0 est maintenant disponible chez votre revendeur habituel.

Pour toutes informations complémentaires, appelez le Service Clients au (1) 69 86 46 46 ou tapez 3616 code Microsoft.

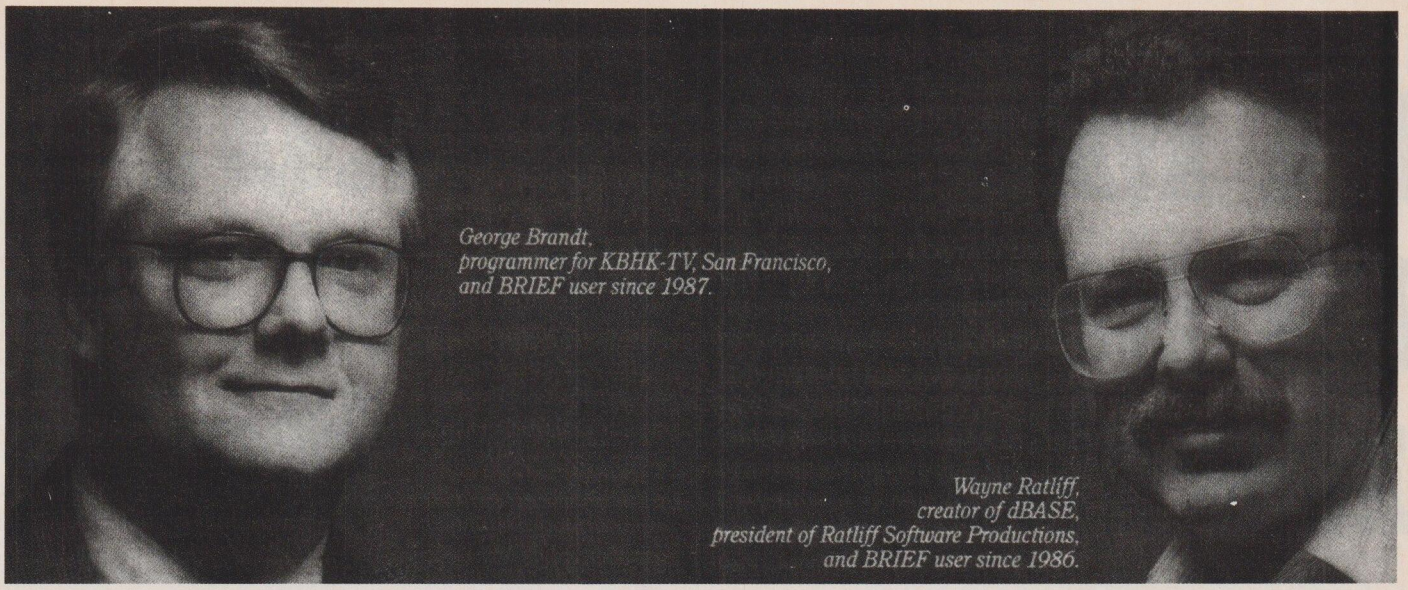
Microsoft, 91957 Les Ulis cedex et à Infomart La Défense - Stand N° 276 niveau 2.

Microsoft®

LES LOGICIELS QUI DONNENT DES AILES.

* Prix public conseillé de la mise à jour au 1/09/91 - i386 est une marque déposée d'Intel Corporation. IBM est une marque déposée d'International Business Machines.

Service lecteurs n° 1 (dernière page)



George Brandt,
programmer for KBHK-TV, San Francisco,
and BRIEF user since 1987.

Wayne Ratliff,
creator of dBASE,
president of Ratliff Software Productions,
and BRIEF user since 1986.

Bien sûr qu'ils ont des besoins de programmation complètement différents, c'est pourquoi ils utilisent

PROMOTION **BRIEF.** ~~3,290 FHT~~ **1490 FHT**

Brief est le seul éditeur de texte orienté programmation du marché à offrir aux développeurs un macro-langage complet, adaptable à pas moins de 52 000 styles différents de programmation.

■ Adaptez Brief à vos besoins

Brief est complètement reconfigurable. Vous pouvez configurer à votre convenance aussi bien le clavier pour créer des macro-commandes, le menu que le macro-langage de type LISP ou C.
Le programme Set Up permet au développeur d'adapter Brief à son compilateur et son style d'édition personnel pour les commandes simples et y adapter le macro-langage aux commandes plus complexes (instructions conditionnelles, boucles, procédures, variables récursives locales ou globales...).

■ LES NOUVEAUTES DE BRIEF 3.0 EN FRANÇAIS

- Avec le nouveau macro-langage de type C, CBrief, les programmeurs en C peuvent écrire des macros rapidement sans passer du C à un autre macro-langage.
- la taille des fichiers a été augmentée à 4 milliards de lignes.
- Le nouveau debogueur de code source fonctionne avec la syntaxe de Brief et le macro-langage CBrief.
- Le nouveau type d'indentation intelligente pour ADA, Cobol, Basic, Fortran, Modula 2, Pascal et C.
- Commandes macros de clavier éditables et sauvegardables.
- Sauvegarde possible des configurations de fenêtrage.

■ LES CARACTERISTIQUES QUI ONT FAIT LA CELEBRITE DE BRIEF

Fonctions complètes de recherche/remplacement, édition de macro-commandes, commande de flexibilité de fenêtrage multiples, commande UNDO des 300 étapes antérieures, taille et nombre des fichiers limités uniquement par l'espace disque, fonction intelligente d'indentation, boîtes de dialogue, compatibilité avec divers modes texte, macro-langage de type LISP sont quelques unes des caractéristiques de l'éditeur de texte le plus complet du marché.

■ Consacrez plus de temps à la programmation

Compilez sans quitter Brief. Le compilateur peut être appelé automatiquement de Brief et se place sur la ligne ou une erreur est décelée en une seule manœuvre.
Grâce aux utilitaires UNDO et REDO, il vous est même permis de faire des erreurs sans grandes conséquences puisque vous pourrez effacer ou récupérer jusqu'à 300 étapes et opérer ainsi toutes les modifications possibles jusqu'à complète satisfaction du résultat.

■ Du fenêtrage facile

Brief supporte 127 lignes et 255 colonnes. Il inclut les standards EGA, VGA, Hercules graphiques plus, Wyse 700, Génius... et possède également une large palette de couleurs.
Il vous suffit de cliquer sur le bouton de la souris pour modifier par exemple la structure complète d'un texte sans passer par le clavier. Vous pouvez aussi ouvrir autant de fenêtres que votre écran puisse en supporter. Vous pouvez sauvegarder, restaurer, changer la taille d'une fenêtre, éditer un même fichier sur plusieurs fenêtres ou déplacer un fichier d'une fenêtre à l'autre.

■ Bâtissez autour de Brief un environnement complet de programmation

Rapide et souple, Brief est accessible à de nombreux autres outils de pogrammation. Ainsi «Solution Systems' Sourcerer's Apprentice», «Charge» ou «Software Performance Analyst» ont été conçu avec un interface transparent à Brief pour efficacité maximale. Brief est compatible avec Microsoft C 6.0 Advisor.
N'oubliez pas dBrief. Il a été conçu pour adapter Brief à un environnement de programmation dBase III+, Nantucket, QuickSilver et Foxbase+.

INNOSOFT

Tél. (1) 40 99 28 00 - Fax (1) 40 99 28 88

DEMANDE DE DOCUMENTATION
BON DE COMMANDE

Nom _____
Société _____
Adresse _____
_____
CP : _____ Tél.: _____

Format de disquettes :
❑ 3,5 pouces ❑ 5,25 pouces

❑ Je souhaite recevoir une documentation sur le produit.

❑ Je commande **BRIEF 3.0** en français au prix de **1838,30 FTTC**
(1490 FHT + 60 FHT de port + 18,6 % TVA)

❑ Je commande **dBRIEF** en version anglaise au prix de **1191,93 FTTC**
(950 FHT + 55 FHT de port + 18,6 % de TVA).

❑ Je joins un chéque du montant de la commande

❑ Je paye par Carte Bleue
N° _____ expirant le _____

❑ Je demande un contre-remboursement
(59,30 fttc de frais supplémentaires).

A retourner à INNOSOFT - 2, rue des Bourets - 92150 Suresnes

Promotion valable jusqu'au 31/12/91

Service lecteurs n° 2 (dernière page)

Sommaire

7
Ressources
Présentation de nouveaux outils de développement sur Mac et PC.

11
Concours GOLD
Forte participation au concours de programmation. Présentation des six gagnants et de leurs travaux.

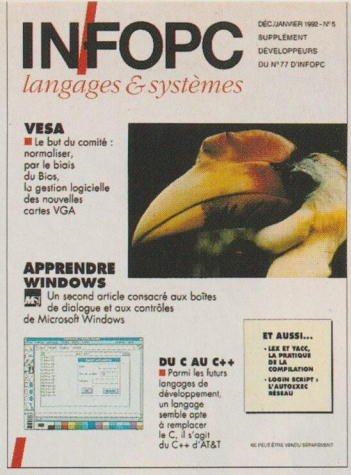
14
Du C au C++ : quels avantages réels ?
La Programmation Orientée Objet est à la mode, et C++ en est l'un de ses fleurons. Qu'apporte-t-il réellement face au langage C traditionnel et quelles sont ses limites ?

24
Lex & Yacc : la pratique de la compilation
Présentation d'outils de programmation complémentaires s'exécutant sous Unix.

34
Les Login Scripts sous Novell
NetWare permet de créer des fichiers de traitement par lots. Ceux-ci s'apparentent aux fichiers batch du DOS en gérant les accès au serveur. Cet article présente les différents types de Login Scripts, la syntaxe de leurs commandes et des exemples de fichiers.

40
Vesa : mode d'emploi
Les cartes super VGA sont légion. Elles offrent toutes des modes graphiques étendus. Le comité Vesa qui regroupe les principaux fabricants de ces cartes a défini des spécifications concernant l'accès par programme aux modes étendus.

50
Apprendre Windows
Cette troisième partie de l'initiation à la programmation sous Windows 3 explique comment intégrer les contrôles et les boîtes de dialogue dans une application.



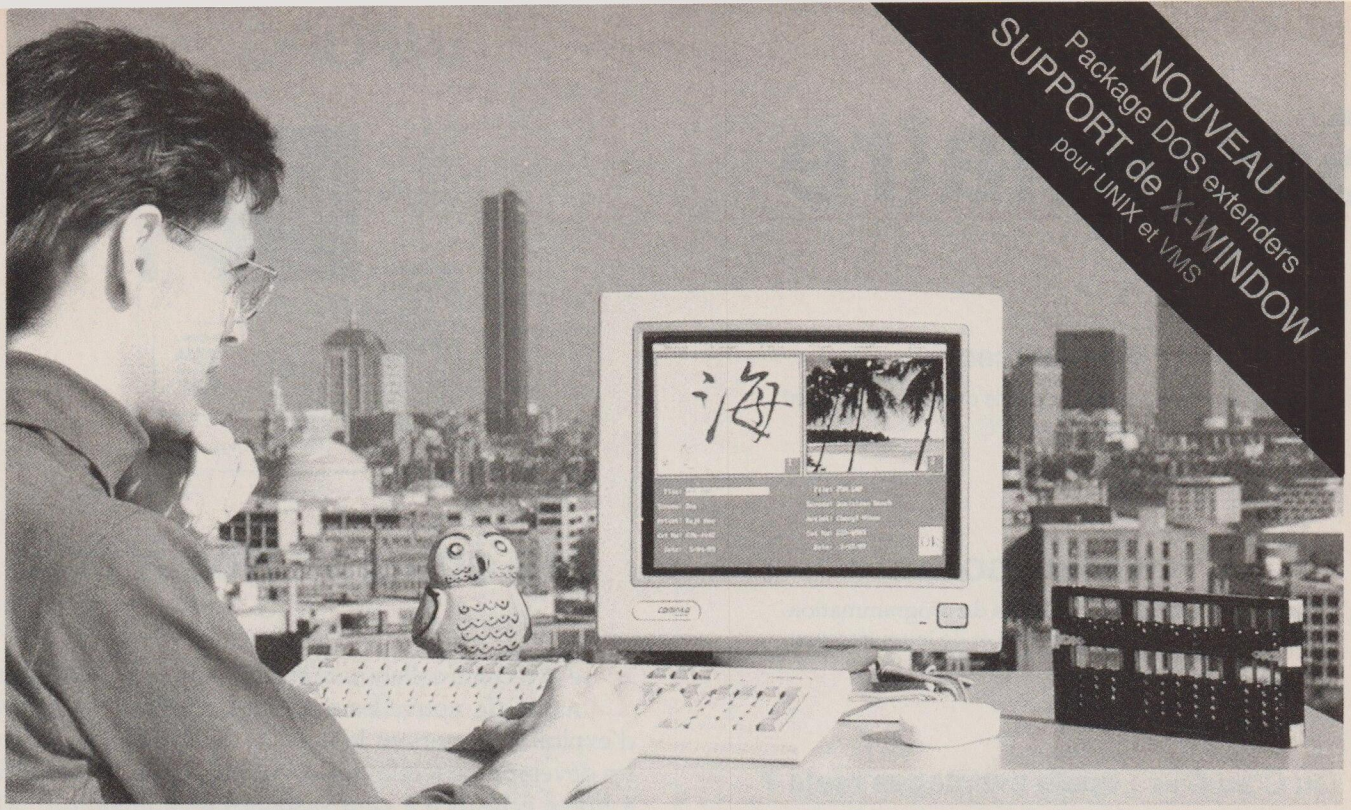
Les éditeurs dos à DOS

De la récente collaboration IBM-Apple devrait bientôt naître de nouvelles machines dotées d'un système d'exploitation portant le nom de code Pink OS/3. Pour les développeurs, cela signifie qu'ils vont être confrontés à brève échéance à cinq systèmes d'exploitation : MS-DOS, OS/2 version 2, Pink OS/3, Unix et enfin Windows-NT. Lequel s'imposera dans cette décennie ? L'échec de la tentative de Microsoft et IBM visant à détrôner MS-DOS avec OS/2, pourtant plus performant, en dit long sur l'inertie du marché. Ce dernier est estimé à 75 millions d'unités équipées de MS-DOS, contre 5 millions de Windows 3.0, 0,6 million d'OS/2 et 1,8 million d'Unix (sources SPA et IDC). Microsoft mise sur Windows. La base installée semble lui donner raison. Toutefois, les machines dotées de cet environnement graphique ne représentent pour l'instant que 7% de la totalité du parc mondial. Il reste donc à convaincre 93% des utilisateurs. Pourquoi franchiraient-ils le pas ?

Ils se verraient obligés au mieux de «gonfler» leur matériel, au pire de le changer, d'acquérir de nouveaux logiciels, sous réserve de leur disponibilité et de leur capacité à exploiter les fichiers créés avec les applications MS-DOS, et de modifier leurs habitudes de travail. De leur côté, les développeurs ont un temps d'adaptation aux outils de programmation pour Windows qui est estimé à environ 6 mois. La plupart des ténors disposant des ressources humaines et financières se sont lancés dans l'aventure. Les autres, les petites et moyennes structures qui écrivent des applications sur mesure, ne leur ont pas encore emboîté le pas.

Des langages faciles à mettre en œuvre tels que Visual Basic et Quick C pour Windows devraient les décider à tenter l'aventure. S'ils le font et s'ils réussissent à convaincre leur clientèle de les suivre, alors Microsoft aura gagné son pari.

Yves Guittard



C-scape : un véritable plaisir.

Gestionnaire d'interfaces écran graphique et texte

Le système de gestion d'écrans C-scape® est une bibliothèque de fonctions en langage C permettant la saisie et la validation de données à l'écran, la gestion de menus et fenêtres, l'édition de texte, et les systèmes d'aide en ligne. L'outil de conception d'écrans de C-Scape, Look & Feel™, vous permet de créer des écrans de façon interactive et de générer le code source correspondant.

C-Scape comprend des fonctions de haut niveau facilement modifiables, aussi bien que des primitives permettant de construire de nouvelles fonctions. Grâce à l'architecture orientée objet de C-Scape, vous développerez des applications plus fonctionnelles, plus flexibles, plus portables, et vous aurez plus de plaisir à le faire.

Un produit phare dans l'industrie du logiciel. Depuis 5 ans, plusieurs milliers de développeurs dans le monde entier se sont convertis à C-Scape, abandonnant leur bibliothèque "maison", ou des produits peu flexibles. La presse approuve: "C-Scape est de loin le meilleur. ...un véritable plaisir", ecrit IEEE Computer. Les plus grandes sociétés ont selectionné C-Scape comme standard de développement.

C-Scape peut être utilisé avec des SGBD, des bibliothèques graphiques, ou toute bibliothèque C ou C++. De plus, C-Scape peut combiner texte et graphique. Pour porter votre application de DOS ou OS/2 vers UNIX, VMS, AIX, recompilez simplement. C-Scape supporte également les "DOS extenders" Phar Lap, Rational et Ergo, ainsi que MetaWINDOW.

Graphique. Fonctionne en couleur, en mode texte ou graphique. Peut lire des images graphiques depuis des fichiers PCX.

Architecture orientée objet. Ajoutez vos propres fonctions et créez des modules de code réutilisables.

Support de la souris complètement intégré pour la selection des options de menu, les champs de saisie, et pour déplacer et redimensionner une fenêtre.

Portabilité. Le code de C-Scape est indépendant du hardware. Supporte DOS, OS/2, UNIX, AIX, VMS, QNX, XENIX... Détecte automatiquement Hercules, CGA, EGA, VGA, et les modes 43 et 50 lignes.

Edition de texte. Comprend un éditeur de texte avec contrôle de la césure, fonctions rechercher/remplacer, et manipulation des blocs de texte.

Flexibilité des champs. Champs masqués, protegés, marqués, avec label, obligatoires, avec ou sans echo. Validation complète des données. Champs de type heure, date, montant, liste déroulante, et autres fonctions de haut niveau. Créez votre type de champ.

Fenêtres. Encadrées, pop-up, superposées, eclatées, ascenceur, défilement horizontal; taille et nombre des fenêtres limités par la memoire RAM uniquement.

Menus. Déroulants, hiérarchiques, pop-up, style Lotus 1-2-3, ou créez le vôtre.

Système d'aide contextuel. Prompts et messages d'aide attachés à un écran ou à un champ. Messages à références croisées pour créer un système d'aide en ligne de style hypertexte.

Gestion des écrans. Ecrans stockés dans des fichiers ou linkés avec l'application.

Génération de code. Créez n'importe quel écran avec l'outil interactif Look & Feel, testez-le, puis convertissez-le automatiquement en code C.

Essayez C-Scape ! C-Scape est *puissant, flexible, portable*, d'apprentissage et d'utilisation facile. C-Scape est livré en standard avec la totalité du code source de la bibliothèque et l'outil de création d'écrans Look & Feel. La documentation est claire, extensive et detaillée. Avec C-scape, pas de module résident et pas de royalties sur les applications diffusées.

C-Scape est un produit d'**Oakland Group**. Smartline assure le support technique de C-scape par téléphone et par fax ainsi que des sessions de formation sur le produit.

Appelez-nous au (1) 46 34 64 19
ou envoyez le bon de commande ci-dessous par fax ou par courrier. Après le plaisir de C-Scape, le développement ne sera jamais plus ce qu'il était.

C-scape et Look & Feel **version 3.2** : DOS, OS/2 (Borland, Microsoft, Zortech): 4500FHT; UNIX: 13500FHT; Autres OS: nous consulter. Cartes de crédit acceptées.

smartline

30, rue Saint André des Arts
75006 Paris France
Fax: (1) 40 46 85 89 ***Tél: (1) 46 34 64 19***

Oakland Group, Inc 675 Massachusetts Ave, Cambridge, MA 02139 USA.Tél: 617-491-7311. Les produits d'Oakland sont distribués en Australie, Belgique, Danemark, Autriche, Allemagne, Suisse, Norvège, Suède, Angleterre, France. La photographie représente un programme C-Scape combinant une saisie de données avec des images vidéo chargées depuis un fichier .PCX. C-Scape et Look & Feel sont des marques déposées par Oakland Group, Inc.; les autres marques citées appartiennent à leurs priopriétaires respectifs. Les fonctionnalités, prix et conditions sont sujets à modification. Documentation en anglais. C-scape (seascape): paysage de mer.

Service lecteurs n° 3 (dernière page)

❑ Je désire recevoir une documentation sur C-scape

❑ Je désire recevoir le dossier d'évaluation comprenant les deux disquettes de démonstration C-scape et Look & Feel. Ci-joint un chèque de 120 F TTC. Une facture sera jointe à l'envoi.

nom prénom _____ société _____ tél _____

adresse _____

Re-sources

Un générateur de systèmes experts L4G

H Expert

• Générateur de systèmes experts L4G • Windows, X-Window, Presentation Manager • Construction, expertise, mondes, classes, objets, attributs, facettes, règles • Explication, raisonnement • Editeur, compilateur incrémental • Moteur d'inférences • Cohérence, intégrité et protection des données •

Le logiciel H Expert est un générateur de systèmes experts sous Windows, X-Window et Presentation Manager, possédant une architecture ouverte sur l'extérieur. Destiné à des applications de diagnostic, de configuration, de planification et d'ordonnancement dans les secteurs de l'industrie, de la banque et de l'assurance, il peut donc reproduire certains mécanismes du raisonnement humain.

La première phase de l'application est la représentation des connaissances qui déterminera la structure de l'expertise. L'utilisateur doit d'abord définir un modèle «classes», «objets», «démons», les attributs et les facettes de classe, l'héritage multiple, la hiérarchie des mondes, les bases de connaissances, et résoudre les éventuels conflits.

Durant la deuxième étape, un moteur d'inférences plein de ressources assure ensuite le chaînage des données, avant, arrière mixte, la détermination des variables, locales ou unifiées, la création et la modification dynamique d'objets et de liens. Les coefficients de vraisemblance et les explications du raisonnement sont également établis. Enfin, H Expert inclu un éditeur interactif et un compilateur incrémental de mondes, classes, objets, attributs, facettes et règles, un module de graphes dynamiques correspondant à ces données. Il gère également la cohérence et l'intégrité lors de la création ou au moment d'une modification. Un système de mot de passe protège les informations. Les fichiers ont un format standard pour une meilleure portabilité vers les principaux SGBD, tableurs et hypermédias, et les échanges dynamiques de données, DDE de Windows, sont utilisés.

L4G, Oracle et SQL

Enfin/2

• Module de développement L4G sous Windows 3 • Presentation Manager et Motif • Architecture client/serveur ouverte • Editeur SQL, langage portable, environnement et débogueur orienté objet • Générateur d'interfaces graphiques • Gestion de projets, modélisation, rapports • DDE, DLL •

Enfin/2 de Enfin Software constitue un environnement de programmation L4G orienté objet, prévu notamment Windows, Presentation Manager, sous OS/2, DOS et bientôt pour Motif sous Unix. Destiné au développement d'applications graphiques pour les utilisateurs finaux, connectable au SGBD Oracle, ce produit se compose d'un «browser» de classes, d'outils de mise au point, inspecteur et débogueur interactifs, et de tout un ensemble de fonctions L4G tel un générateur d'interfaces graphiques incluant la gestion par icone avec multifenêtrage, un système de rapports, un éditeur SQL, une zone de requête et d'autres facilités pour la modélisation, la définition et la modification des tables. Moyennant quelques adaptations minimes, les applications et les fichiers sont d'une grande portabilité. Enfin/2 a une architecture ouverte qui supporte les échanges dynamiques de données et de bibliothèques.

Générateur et outils en C++

Bashlib++

• Bibliothèque objet C++ • Générateur d'applications sous DOS avec transparence du matériel • Construction système de fichiers, fonctions de gestion mémoire, écran, clavier, souris, imprimante, base de données relationnelle, multitâche et émulation pour connexion réseau •

Bashlib++ de Bashar est une bibliothèque de fonctions évoluées en C++ conçue pour tout type de configuration PC sous DOS. Réduisant le temps d'analyse, de programmation et de maintenance, constitue également un générateur d'applications du type gestion des stocks, comptable, commerciale, tableur ou émulation réseaux, imposant toute une méthode de travail.

Dans un environnement graphique et convivial, l'utilisateur pour concevoir un logiciel doit d'abord préciser la structure des fichiers, leurs interactions potentielles et les principaux événements. Ensuite, l'interface utilisateur, les menus, les masques de saisie, les touches raccourcis et la gestion de l'impression se trouvent définis aux différents stades de l'exécution.

Ce produit permet également de récupérer des bases de données relationnelles conçues par Paradox ou d'en créer avec l'outil PXUtil livré avec Bashlib++, de s'y déplacer ou d'insérer des modifications. Il existe donc de nombreuses fonctions pour construire les menus, les sous-menus, le multifenêtrage, les zones de saisie et utiliser la souris.

Les sorties temporaires sous DOS, l'allocation mémoire, les co-routines multitâches, le style et le corps de l'impression plus simplement accessibles.

Un kit de développement graphique pour les PC

Summer Windows est un serveur graphique polyvalent et multistandard, fonctionnant sous DOS. Connectable aux Turbo Pascal et C, Quick Basic et MCS, il fournit à ces compilateurs près de 180 primitives et de nombreux outils.

Le moteur d'objets interactifs gère l'émission des messages sur requête d'une application et prend entièrement en charge les opérations de déplacement et de réaffichage de ces objets. L'éditeur d'interfaces met à disposition des programmeurs, pour la mise au point et la maintenance de l'aspect des applications, des ressources graphiques, menus, fenêtres, aide en ligne, d'une utilisation simple et transparente.

Le Composant Logiciel, format défini par Summer Win-

Re-sources

Summer Windows

• Serveur graphique polyvalent pour PC sous DOS • Connectable T. Pascal et C, Quick Basic et MCS • Gestion multilangage et multitâche. Moteur d'objets interactifs • Editeur d'interfaces • Didacticiel, exemples, commentaires • 32 Mo mémoire paginée applications •

dows, supporte la création de programmes multilangages et multitâches, sans limite de taille, pouvant implémenter une bibliothèque dynamique, se connecter ou piloter un menu et gérer une aide en ligne hypertexte. L'éditeur du logiciel donne de nombreux exemples de programmes commentés pour tous les langages.

Il existe deux versions du logiciel, SW et SW 386. Cette dernière n'occupe que 50 Ko en mémoire, peut créer jusqu'à 32 Mo de mémoire paginée pour les applications et supporte les modes d'affichage, d'EGA à VGA 256 couleurs.

Un débogueur furtif

Le débogueur Périscope IV répond aux principaux problèmes rencontrés par les professionnels durant la mise au point des programmes et se revèle moins coûteux que les ICE. Assisté par une carte spéciale, il n'a plus d'influence sur la vitesse d'exécution du programme traité et n'utilise plus la mémoire de l'unité centrale.

Particulièrement intéressant pour les applications en temps réel et de communication sous DOS, le périscope ne fonctionne que sur les AT 386, cadencés au maximum à 33 MHz. Les adaptations à Windows, à OS/2 et aux autres configurations matérielles sont en cours de développement.

Grâce à un «trace buffer» de 16 Ko, les informations s'obtiennent directement du microprocesseur, ce qui assure une totale indépendance lors de la définition et du contrôle des points d'arrêt sur les événements du CPU, lesquels prennent trois formes : en mémoire avec lecture, écriture, encadrement du code, reconnaissance d'interruptions sur les données de type octet, mot simple ou double; port série.

Le «pass counter» stoppe l'exécution d'un programme après un nombre de points d'arrêt spécifié, et le «sequential trigger» permet de visualiser des événements durant une séquence précise. Opérationnel à tout moment, ce débogueur trouvera ainsi des erreurs que les autres ne pouvaient déceler. Le fichier de capture de ces informations peut avoir trois formats d'affichage. Le format «raw dump» montre l'adresse, le contenu, le statut des données et un compteur d'enregistrement.

Le format «disassembly», en utilisant les cycles «prefetch», n'affiche que les instructions transmises. Enfin, le format «combination», comme son nom l'indique, est un mélange d'instructions et de données.

Périscope IV

• Débogueur indépendant • Carte avec RAM de 2 à 16 Ko • Applications temps réel et communication • DOS AT 386 33 MHz au maximum • Points d'arrêt sur mémoire, données, port série • Format de capture: dump, désassemblé et combiné •

Le multitâche en Pascal...

MTASK Professionnel est une bibliothèque de fonctions permettant de réaliser, en Turbo Pascal sous DOS, des applications multitâches et en temps réel. Pour l'acquisition et l'exploitation des données d'une expérience en laboratoire, la robotique, la télématique, l'animation simultanée de plusieurs graphiques, ce module propose une centaine de primitives.

La gestion des tâches apparaît à la fois puissante, souple et simplifiée : ordre de priorité, commutation, création, destruction, arrêt temporaire et timer par interaction. La taille de la pile par tâches est paramétrable, le nombre de tâches ne connaît pas de limite au niveau logique. En n'occupant que 10 Ko en mémoire vive, MTASK gère les environnement graphiques et le texte, les indicateurs de type sémaphore, les attentes d'interruption ou d'affectation de valeurs à une variable, et aussi les entrées/sorties sur les voies de communication série, jusqu'à 9 ports. On accède directement aux primitives, car une dizaine d'entre elles sont livrées avec les fichiers sources.

Aide «on line» sur Macintosh

Symantec a mis au point, dans la grande famille des Think, un nouvel outil de référence «on line» pour les programmeurs sur Macintosh.

Pourquoi ? Les professionnels du développement perdent souvent beaucoup de temps à se documenter et à rechercher des informations dans les nombreux manuels

MTASK Professionnel

• Bibliothèque de fonctions multitâche et temps réel • Turbo Pascal sous DOS • Occupe 10 Ko en RAM • Gestion des tâches souple et simplifiée. Création, attente, destruction et déclenchement interactifs • Gestions, interruptions, et affectations, sémaphores • 90 instructions, 10 avec sources. Communication 9 ports série • Utilitaires •

sur le système et sur la machine exploitée. Think Reference représente un concentré des volumes I à IV d'Inside Macintosh, accessible à tout moment par une simple commande du clavier. Les utilisateurs des Think C et Pascal ou de MPW trouveront dans ce produit une description complète des routines de la boîte à outils, des notes techniques, ainsi que de nombreux exemples et conseils pratiques. Les polices de caractères, les ressources, bref tous l'univers Macintosh s'y trouve exposé, avec d'ailleurs plusieurs illustrations.

Le module d'aide magistrale et contextuelle sera très apprécié pour sa souplesse d'utilisation autant que pour ses vertues pédagogiques. Sous forme d'hypertexte, les données peuvent être consultées selon plusieurs modes de recherche, par trappes, gestionnaires, mots-clés, interfaces et structures.

En plus d'une telle indexation, un système de repères de pages donne la possibilité aux programmeurs de personnaliser le produit, suivant leurs besoins spécifiques et la fréquence des informations. Enfin, des modèles de fonctions spéciaux permettent de copier les appels de la ToolBox dans le programme.

Re-sources

Microsoft Source Profiler

Le Source Profiler de Microsoft est un outil permettant l'analyse des performances des applications développées pour les environnements DOS, OS/2 ou Windows. S'adressant aux développeurs professionnels travaillant en C, en macro-assembleur, en Basic, en Fortran ou en Cobol, ce produit détecte les portions de code devant être optimisées ou corrigées selon une méthode prédéfinie. Ce profileur génère quatre types d'informations lors du diagnostic effectué sur des lignes ou des

Microsoft PDS

• Profiler Analyse des applications DOS, OS/2 ou Windows • Compatible avec les langages C, macro-assembleur, Basic, Fortran et Cobol de Microsoft • Détection du code à optimiser ou à corriger • Contrôle du temps CPU, décompte et repérage des portions utilisées ou non • Comparaison relative du temps consommé par symbôles • Indépendance de l'analyse, fusion, cumul, exportation des résultats • Analyse et manipulation des données à partir d'un graphique • Palette de graphes en 2 ou 3D, et de tous types • Solveur d'équation, fonctions statistiques et mathématiques • Fonctions de lissage, régression et interpolation • Mode Commande ou intégration dans les systèmes PDS •

fonctions que l'utilisateur peut déterminer au cours de la réalisation du programme. D'abord, pour mieux connaître la répartition du temps CPU, le temps d'exécution de chaque fonction est indiqué. Ensuite, le Source Profiler compte le nombre de passages dans les diverses boucles et les appels de routine.

Parallèlement, le logiciel marque avec des indicateurs binaires les branches inertes de l'application ou celles qui n'ont pas été encore testées. Enfin, le programme fournit une comparaison relative du temps consommé par chacun des symboles du programme. Ces quatre mesures peuvent se combiner au gré des besoins de l'utilisateur et sont totalement indépendantes les uns des autres. Ainsi, il devient possible de fusionner ou de cumuler les résultats obtenus, par ailleurs exportables vers Excel ou un traitement de texte, pour analyse complémentaire ou présentation plus élaborée. Exécutable en mode Commande, le Source Profiler peut également s'intégrer dans le «Programmer's WorkBench» des systèmes de développement de la famille

Un Fortran 5.1 pour Windows

Cette nouvelle version du célèbre compilateur PDS de Microsoft se distingue surtout par une amélioration de la gestion mémoire, l'intégration des outils de développement dans le Programmer's WorkBench ainsi que l'interfaçage avec les langages C ou Visual Basic, afin de créer de véritables applications Windows.

Microsoft Fortran permet aux programmeurs d'exécuter sous Windows les applications développées précédemment en Fortran 16 bits, simplement en les recompilant avec les spécificités de cet intégrateur graphique. Pour cela, une nouvelle bibliothèque QuickWin est mise à disposition afin de faciliter l'édition des liens et l'intégration des applications DOS sous Windows.

En effet, les développeurs

Microsoft Fortran

• Plate-forme de développement intégrée • Gestion mémoire optimisée, jusqu'à 64 Mo en mode virtuel sur un AT 286 • Support de Windows : bibliothèque • QuickWin, création de DLL • Compatibilité ascendante code Fortran 16 bits • Conformité norme ANSI 77, extension VAX, IBM •

bénéficieront de fonctionnalités avancées telles que l'ajout de titres aux fenêtres, la gestion des boîtes de messages, la définition de fenêtre-fille, etc. En fournissant cet accès à Windows, ce compilateur s'exécutera sur un AT 286, avec une optimisation de la mémoire virtuelle, jusqu'à 64 Mo. Comme avec les autres plates-formes de programmation professionnelles de la gamme Microsoft PDS (C,Basic, Cobol et Macro-Assembleur), les outils nécessaires au développement sont regroupés dans un même environnement, le Programmer's WorkBench.

L'utilisateur y contrôle toutes les phases de la conception d'un logiciel : saisie du code source, accès à l'aide en ligne, compilation, édition des liens, déboguage et gestion des références croisées. Ce compilateur peut également créer des bibliothèques dynamiques (DLL) , pour réaliser des applications en langages mixtes tirant pleinement profit de l'environnement Windows.

Il est possible d'appeler ces routines Fortran par un logiciel en C ou Visual Basic. Ces bibliothèques gèrent aussi les codes en Fortran standard et les fonctions déjà existantes dans les applications Windows. Totalement conforme avec la norme Fortran et ANSI 77, et compatible avec les extensions VAX et IBM, le Microsoft Fortran assure la portabilité d'applications mini ou grand système sur PC. Hors mis les avantages du multitâche en mode 386 étendu et la disponibilité effective de nombreux outils d'analyse, ce produit devrait séduire les habitués du langage Fortran.

L'analyse "look and feel"

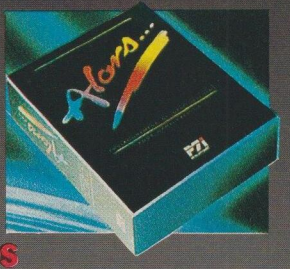
Delf commercialise depuis peu une version 3.0 de Système Delf, totalement réécrite pour s'intégrer à l'environnement Windows, tout en tenant compte des normes SQL et des récentes évolutions de la méthode d'analyse Merise. Cet outils a pour objet de gérer le système d'information, de définir les spécifications, et de produire la documentation correspondante. Les concepteurs du logiciel, Merisiens convaincus, s'appuient sur le standard de Windows, qui facilite la communication entre les applications et la manipulation des données.

Il bénéficie aussi d'une approche graphique et d'un gain de convivialité évident. Grâce à l'utilisation du langage SQL Windows, développé par Gupta Technology, Système Delf offre une relative indépendance vis-à-vis des gestionnaires de données, ainsi que la possibilité de fonctionner sur un plus grand nombre de plates-formes. Ce système est compatible avec B Trieve, SQL Base, SQL Server, Oracle, Informix, les serveur DOS et Unix avec des passerelles mini ou mainframe. Les dernières évolutions de Merise sont prises en compte.

Alors... MD

UN AGL POUR NANTUCKET 5

- Dictionnaire de données
- Générateurs de sources
- Bibliothèque



La maîtrise du temps

Alors... intègre trois modules complémentaires : un dictionnaire de données, des générateurs de sources et une bibliothèque, destinés à aider le développeur à gagner du temps et créer des sources très lisibles.
A l'image des moteurs relationnels, le dictionnaire de données permet de créer les fichiers, les index, les contrôles, les liens interfichiers et d'en garantir la cohérence hors des lignes de programmes.

Un très petit nombre de lignes à maintenir

Le dictionnaire élimine toute l'écriture des contrôles interfichiers. La bibliothèque comporte une cinquantaine de fonctions majeures pour les développeurs (attribution d'un nom de fichier temporaire en réseau, log d'erreur automatique, recherche d'un champ dans un autre fichier...). Les générateurs développent un ensemble de fonctions modulaires (entrées/sorties, menus déroulants, doc. technique...).

Une maintenance aisée

Voici quelques exemples :
Un champ change de longueur? Tous les éléments utilisant ce champ fichiers, index, écrans et variables mémoires sont modifiés en cascade et reconstitués dynamiquement.
Un index est nécessaire pour un nouvel état? Vous le déclarez dans le dictionnaire et tous les programmes déjà écrits le reconnaissent immédiatement et le prennent en compte.
Pour en savoir plus renvoyez-nous le coupon ci-dessous.

PTI DEVELOPPEMENT

88 avenue Kléber - 75116 Paris
Tél. 47.27.01.29 ■ Fax 47.04.66.08

Versions 5 et 10 postes disponibles ■ Version monoposte 15 000 F HT

Je souhaiterais ❑ Une documentation.
❑ Le manuel de découverte avec la disquette de démonstration. Ci-joint un chèque de 177,90 F TTC.
❑ Assister à une démonstration gratuite.

Nom _____ Fonction _____
Société _____ Tél. _____
Adresse _____

Nantucket est une marque déposée par Nantucket Corporation

Service lecteurs nº 4 (dernière page)

Re-sources

Système Delf 3.0

• Génie logiciel basé sur Merise • Totalement ré-écrit pour Windows • Dernières évolutions de la méthode d'analyse • Améliore la communication entre les applications et manipulation des données • Langage de requête SQL Windows • Compatibilité avec de nombreuses plates-formes, serveurs DOS, passerelles Unix • Participation des utilisateurs à l'amélioration logicielle •

L'utilisateur pourra gérer les objets types, sous-types, les contraintes entre objets et celles entre relations. De plus la méthode Merise pourra bientôt inclure les traitements concernant les liens entre données et processus, les opérations, procédures, phases et la mise en évidence de primitives d'action. Les spécifications décrites par les concepteurs seront alors plus précises. Précisons que la société Delf laisse la possibilité aux utilisateurs de participer dans des groupes de travail à l'amélioration du logiciel.

Réseaux

La version 2.1C de Supervise fournit un ensemble d'outils d'aide à l'administration et à la gestion du réseau Novell Netware. Commercialisé par NSE, ce logiciel regroupe dans une même interface les utilitaires Syscon Filer, Fconsole, Pconsole, Printcon, Printdef, Makeuser et Volinfo. Tout en respectant les niveaux de sécurité attribués sur le réseau, Supervise permet aux utilisateurs de personnaliser leur environnement de travail. Ce logiciel interagit en temps réel avec la "Bindery" des serveurs Novell, et les différents fichiers systèmes du réseaux. Ainsi, il donne accès aux procédures de login, au gestionnaire d'impression, au journal des erreurs et autres fichiers de description. L'administrateur peut aussi manipuler la structure des répertoires de connexion et les arborescences liées à chaque adaptateur de réseau, avec notamment la prise en compte de la variable "%P_station. De plus, Supervise intègre deux puissantes fonctions. L'une permet la création de 100 utilisateurs simultanément, l'autre assure les copies étendues à travers la structure des répertoires. Pour ce qui est de l'impression, ce produit offre deux principaux outils. Le premier, Supervise permet d'imprimer la totalité des données de configuration du réseau, selon de nombreux critères de sélection. Le deuxième détermine si une tâche, ou un format d'impression est propre à un utilisateur, ou si elle s'apparente au groupe. Pour mieux maîtriser toutes ces fonctionnalités, une aide en ligne, de type hypertexte, est en permanence dis-ponible durant le traitement. NSE distribue aussi MENU-NSE, outils complémentaire de Supervise, pour la gestion des menus.

SUPERVISE 2.1 C

Utilitaires de méthodologie et d'administration des réseaux Novell • Gestion des niveaux de sécurité • Personnalisation de l'interface et de l'impression pour les utilisateur • Interaction en temps réel et accès permanent à la "Bindery" et aux fichiers systèmes • Maintenance des répertoires et arborescence • Fichiers de procédures pour la création d'utilisateur et les copies étendues • Analyse et traitement de la configuration réseau • Version mono ou multiposte •

Matthieu Thfoin

CONCOURS GOLD : les gagnants

Le Grand Open Logiciel et Développement a pour but de révéler de nouveaux talents. Il est ouvert à toute personne ayant conçu une application professionnelle et se décline en deux catégories. La première s'adresse aux programmes développés au moyen d'un langage ou d'un compilateur de type C, Pascal, Basic, etc. La seconde regroupe les programmes développés par des générateurs d'applications commercialisés sous forme de package de type Windows Toolbook, Clipper, Foxpro, Paradox, etc. Trois prix sont décernés dans chacune des trois catégories en fonction de critères de sélection qui portent sur l'intérêt du sujet pour les utilisateurs, l'originalité de l'application, sa finition, sa convivialité, ainsi que sur la qualité de la documentation.

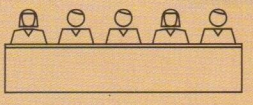
54 PROGRAMMES AU BANC D'ESSAI

Le GOLD 91 a été un succès. Nous avons, en effet, reçu 54 programmes, dont 33 pour la première catégorie. La plupart d'entre eux présentaient une réalisation digne d'éloges, à tel point que nous avons eu du mal à les départager.

Le CNIT

La remise des prix s'est déroulée au Salon Software Development qui s'est tenu au CNIT du 15 au 17 octobre. Celui-ci, en regroupant les principaux intervenants du marché logiciel, est devenu le lieu de rencontre privilégié des utilisateurs et des développeurs.

LE JURY



Le jury se composait de M. Olivier EZRATTY, chef de groupe de Langages et Outils de Développement de Microsoft France, de M. Patrice PRUT, président de la société PC Soft, de M. Philippe RODRIGUEZ, chef de produit de la société Top Log, et d'Yves GUITTARD de la revue Langages & Systèmes. La société Hewlett-Packard a également contribué en nous prêtant les machines sur lesquelles ont été installés tous les programmes.

Logiciels de première catégorie

1er prix

M. Philippe GOUEFFON
pour Alternate Forth
Cette variation du langage Forth a été réalisée sous Basic PDS 7.0.
M. GOUEFFON a gagné un micro-ordinateur HP Vectra 386 16 N, un Magic II de Top Log, un High Screen 5 de PC Soft, la version télématique par VIF de son logiciel et le langage Visual Basic de Microsoft.

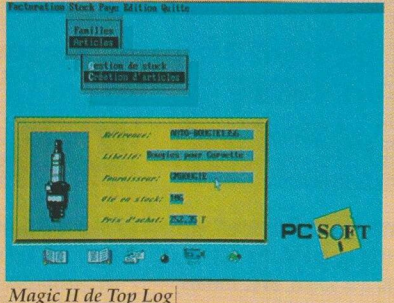
2ème prix

M. Emile RABIN
pour Screen L++
Ecrit sous Turbo Pascal 5.5, Screen L++ est un outil de développement d'application en Pascal. Il permet de définir l'interface utilisateur de l'application, y compris l'enchaînement entre les différents menus et la grille de saisie.
M. RABIN a gagné un Visual Basic de Microsoft, un package de développement de Microsoft et un High Screen 5 de PC Soft.

3ème prix

M. Gérard BAUER
pour Gestcompte
Ce logiciel de gestion sous Windows assure le suivi et le contrôle de comptes (au nombre illimité) vis-à-vis de la banque et du carnet de chèques. Gestcompte a été écrit à l'aide de Zortech C++, C++/View et SDK Windows 3.
M. BAUER a gagné un Visual Basic de Microsoft et un package de développement de Microsoft.

HP Vectra 386 ↑ *High Screen 5 de PC Soft* ↓



Magic II de Top Log ↓

Logiciels de seconde catégorie

1er prix

M. Jean-François FALOVIEZ
pour Suipi
Suipi est un logiciel de suivi des relations publiques développé sous Magic II.
M. FALOVIEZ a gagné un micro-ordinateur HP Vectra 386 16 N, un Magic II de Top Log, un High Screen 5 de PC Soft, la version télématique par VIF de son logiciel et le langage Visual Basic de Microsoft.

2ème prix

La société MICROLOG
pour MC-Winner II
Développé sous Polymode, MC-Winner II est un logiciel de gestion de prospects.
La société MICROLOG a gagné un Visual Basic de Microsoft, un package de développement de Microsoft et un High Screen 5 de PC Soft.

3ème prix

M. Bernard GISSLER
pour Messagerie Vocale
Développé à l'aide de l'hyper pack de PC Soft, Messagerie Vocale est, comme son nom l'indique, un logiciel de messagerie vocale pour PC.
M. GISSLER a gagné un Visual Basic de Microsoft et un package de développement de Microsoft.

DEVELOP

HIGH SCREEN 5

Générateur d'écrans

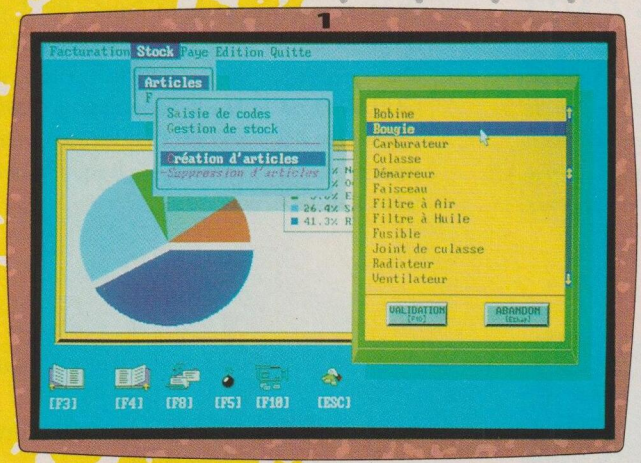
- DOS, WINDOWS, OS/2, UNIX.
- Tous langages (oui, tous!)

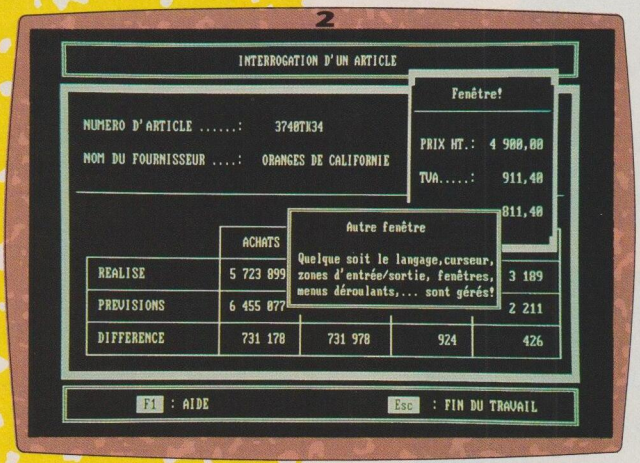
1- Menus déroulants, fenêtres, saisies, icônes, graphiques,... : tout High Screen 5 en un écran !

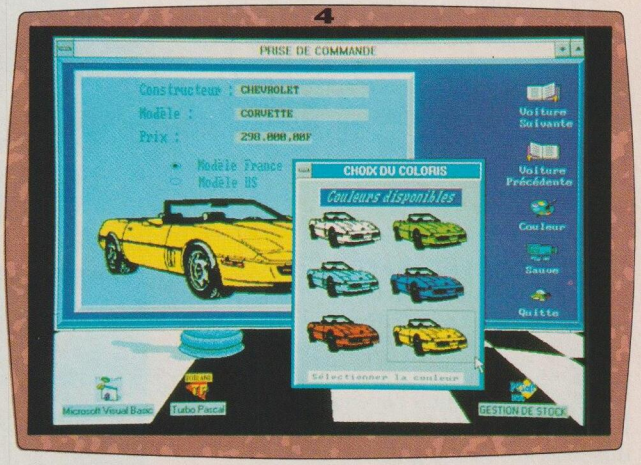
2- High Screen fonctionne sur tous les écrans...

3- Une application graphique sous DOS...

4-... la même sous Windows (compatibilité !)







Que vos applications soient en mode texte ou graphique, au clavier ou à la souris, avec menus déroulants, avec des icones, avec ou sans fenêtres,
High Screen 5 vous permettra de développer 3 à 10 fois plus vite.

La programmation est très simple, avec des ordres en français.
De nombreux utilitaires sont livrés : éditeur d'icones, de dossiers, maquettage,... Tout devient possible, dans tous les langages!
(C, Basic, Pascal, dBase, Nantucket, Cobol, Fortran,...)

High Screen 5 a déjà changé la vie de milliers de développeurs : essayez-le !

A partir de
4900 F HT
5 811,40 F TTC

Disquette de démonstration disponible : appelez !

Nos produits ont reçu de nombreuses récompenses :

HYPER PACK — OUTILS DE DEVELOPPEMENT — PRIX D'EXCELLENCE — micro SYSTEMES

Meilleur outil de développement — SOFT & MICRO

LABO QUALIFIE — INFO PC

LE SUPPORT TECHNIQUE
Un des points forts de PC SOFT est le support technique, efficace, disponible et... sympathique.
Notre équipe de techniciens chevronnés est à votre disposition. Gratuitement.

GARANTIE "ESSAI SANS RISQUE"
Si le produit ne vous convenait pas pour une raison quelconque, vous pourriez nous le retourner (en état d'origine bien sûr) dans les 10 jours suivant sa réception pour un remboursement intégral. Conditions au dos du tarif.

Malgré le soin apporté à la rédaction de ce document, les informations ne sont pas contractuelles. Toutes les marques sont déposées par leur propriétaire respectif.

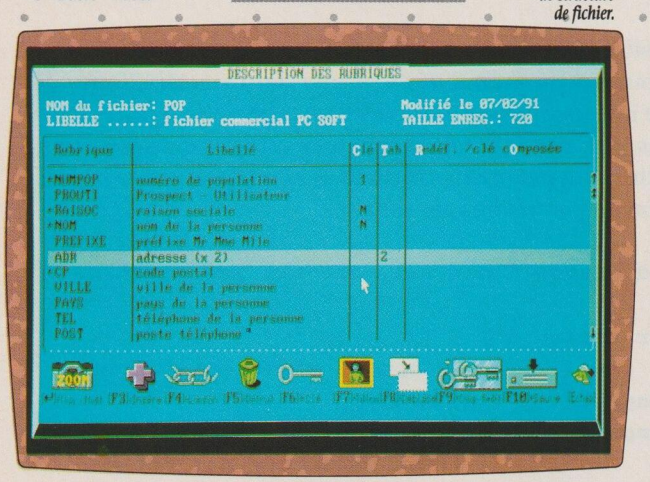
PEZ VITE

HYPER FILE 3

SGBD Gestionnaire de fichiers

- DOS - WINDOWS - OS/2
- C - Basic - Pascal

Création de structure de fichier.



Hyper File 3 est le plus rapide des SGBD. Après avoir défini facilement la structure des fichiers, leur utilisation dans les programmes est très simple.
La programmation (en C, Basic, Pascal) s'effectue en français.
De plus Hyper File 3 édite automatiquement le dossier d'analyse.
De nombreux utilitaires sont livrés.

Si vous manipulez des fichiers, testez Hyper File !

A partir de
4900 F HT
5 811,40 F TTC

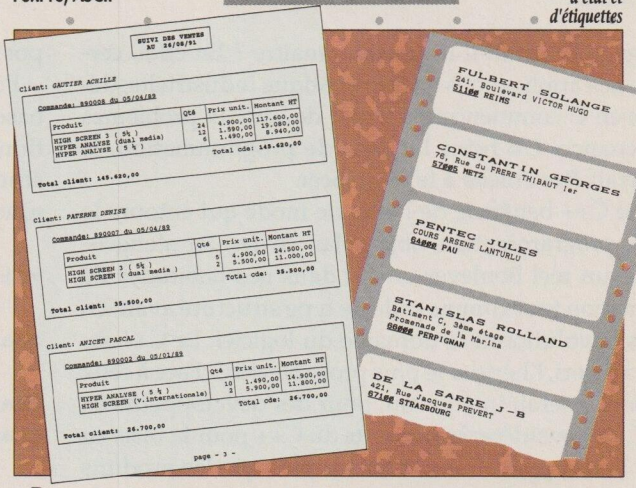
Disquette de démonstration disponible : appelez !

HYPER PRINT 3

Générateurs d'états et d'interrogations

- C, Basic, Pascal, dBase, Nantucket, FoxPro, ASCII

Exemple d'état et d'étiquettes



Réaliser des états imprimés de gestion en quelques minutes seulement, c'est possible avec Hyper Print 3.
Hyper Print 3 rapproche jusqu'à 9 fichiers par état et sait calculer. Le résultat s'édite sur imprimante (matricielle ou laser) ou à l'écran, ou même dans une feuille de tableur.

Un produit à évaluer d'urgence !

A partir de
4900 F HT
5 811,40 F TTC

Disquette de démonstration disponible : appelez !

HYPER PACK DEVELOPPEUR

Atelier de Génie Logiciel

C - Basic - Pascal

Ce "pack" regroupe High Screen, Hyper File et Hyper Print qui utilisent le même dictionnaire de données.
Le "PACK" permet de développer de superbes applications rapides, très très vite !

A partir de
9900 F HT
11741,40 F TTC

PC SOFT est Fournisseur Officiel de la Préparation Olympique

PC SOFT

LIVRAISON RAPIDE
En France métropolitaine, livraison en 24 heures par transporteur.

~~DISQUETTE~~ DE DEMONSTRATION DISPONIBLE

DOCUMENTATIONS CLAIRES EN FRANCAIS

PAS DE REDEVANCES

SIEGE MONTPELLIER
Tél. (16) 67.032.032
216, rue des Escarceliers, BP 3019
34034 Montpellier Cedex 01
Fax: (16) 67.03.07.87
Support Technique: (16) 67.03.17.17

PARIS
Tél. (1) 48.01.48.88
34, Boulevard Haussmann
75009 Paris
Télex: 290.266 F (MBI)
Minitel: 3614 code PCSOFT

DÉCOUVERTE

DU C VERS LE C++ :

Quels avantages réels ?

ALAIN DORSEUIL

Le langage C aurait-il trouvé son maître ? En effet, certaines des applications en informatique industrielle, développées en C commencent à poser de sérieux problèmes de maintenance. Un futur langage de programmation, le C++, pourrait être amené à le remplacer.

Certes, le C++ bénéficie de l'effet de mode qui entoure le concept de programmation objet. Mais ce dernier n'engendre-t-il pas un réel bouleversement dans nos habitudes de programmation tout comme celle de type structuré avait, en son temps, révolutionné le domaine du logiciel. Le C++ serait donc, a priori, l'héritier le plus probable du C. Face à cette constatation, il serait intéressant, pour des développeurs en langage C, de considérer les apports du C++ pour la mise en œuvre de leurs applications; d'autant plus que les produits (chaînes de compilation, générateurs de classes, outils d'aide à la programmation par objets, etc.) existent sur le marché, et ce aussi bien en environnement DOS que sous Unix.

Offrir un langage facile à maintenir sans pour autant perdre les qualités de compacité et d'efficacité du langage C, tel se présentait l'un des objectifs de Bjarne Stroustrup lorsqu'il entreprit le développement du C++ au début des années 80. Pour bien cerner l'intérêt de C++ par rapport au C, il convient de distinguer trois points essentiels qui nous serviront par ailleurs de trame pour le présent article :

- C++, c'est avant tout du C amélioré;
- C++ supporte les concepts d'abstraction (le premier «plus»);
- C++ supporte les concepts de programmation par objets (le second «plus»).

C++, le C en mieux

Depuis son apparition en 1970, le langage C (au standard K&R) s'est sensiblement enrichi de nouvelles fonctionnalités (et débarrassé de quelques défauts) grâce à la norme définie en 1983 par l'ANSI. Celle-ci, respectée par la majorité des compilateurs actuels, a partiellement tiré parti des spécifications de base du C++ dont, par exemple, la possibilité de définir des entités de types «const» et «enum».

Pour un programmeur C, l'une des caractéristiques de base intéressantes en C++ concerne le prototypage des fonctions qui, outre le fait d'effectuer une comparaison plus stricte entre paramètres formels et effectifs, autorise la définition de paramètres par défaut :

```
void change(float val, int tx, int code=25,

char sigle='$') { //... };
                // appels effectifs corrects :
change(12.4,6,12,'£');
change(58.3,8,17);
                // un paramètre par défaut
change(34.5,7);
                // deux paramètres par défaut
```

Trois autres aspects de base pourront concerner le programmeur C : l'attribut «inline», le type «reference» et le principe d'allocation dynamique de mémoire.

L'attribut «inline» autorise une nouvelle forme de définition de macrofonctions paramétrées. Tout programmeur C averti connaît les effets de bords que peut provoquer une définition du type :

```
#define SOMME(i,j)  i+j
/* sans les parenthèses... */
```

Le problème principal vient du non-typage des macrofonctions définies par le préprocesseur, ce dernier n'effectuant qu'une substitution de chaîne. En C++, une telle macrofonction est obligatoirement typée (ici, de type entier) et se déclare comme suit :

```
inline int somme(int i,int j) { return i+j };
```

Contrairement au «#define», l'attribut «inline» préserve la sémantique de l'appel. Dès lors, le compilateur effectue une vérification de type et, selon le principe classique d'une macro, étend le corps de la fonction pendant des appels successifs.

Si la majorité des applications en informatique industrielle ont été développées en langage C depuis plus de quinze ans, actuellement certaines d'entre elles commencent à poser de sérieux problèmes de maintenance, tant corrective qu'évolutive. Parmi les futurs langages de développement, et à l'image de ce qui se passe outre-Atlantique, il semblerait que le langage C++ d'AT&T soit le plus apte à remplacer le C.

Le type «reference» en C++, déclaré par l'opérateur «&», permet de corriger quelques défauts inhérents à la notion de pointeur en C. D'une part, une déclaration de référence ne peut se faire sans initialisation :

```
int valeur=88;
int &ref_valeur;            // Erreur
                            // de compilation
int &ref_valeur=valeur;     // OK

printf(«%d»,ref_valeur);    // Affiche 88
valeur=22;

printf(«%d»,ref_valeur);    // Affiche 22
```

D'autre part, une «reference» autorise un style d'écriture plus homogène pour le prototypage d'une fonction dont les paramètres sont passés par adresse (à condition d'admettre qu'un programmeur écrira plus naturellement «x» que «*x» pour faire référence à une variable x).

```
// Style C
void decremente(int *x)
{

*x=*x-1; //C... };
main()
{
int valeur=9;
int pt_valeur;
pt_valeur= &valeur;
decremente(pt_valeur);    //C...
}

// Style C++
void decremente(int &x)
{
x=x-1;                    // C++ };
main()
{
int valeur=9;
decremente(valeur);       // C++ }
```

La difficulté en C vient en partie du fait que tout passage de paramètres s'effectue par valeur, le passage par adresse obligeant à déclarer un «pointeur sur» (ptr) et à référencer la variable indirectement par le pointeur (*ptr). Notons pour terminer que le type «reference» permet, en cas de nécessité, de disposer facilement d'un alias pour une variable.

En C++, tout comme en C, on peut toujours gérer dynamiquement la mémoire à l'aide des fonctions de bibliothèque telles que Malloc() et Free(). Néanmoins, par extension, cette gestion peut être réalisée en C++ par deux opérateurs «new» et «delete» respectivement responsables de l'allocation et de la désallocation. L'opérateur «new» autorise, par exemple, d'initialiser un pointeur sur un type d'objet quelconque contrairement à la fonction Malloc () qui, en C-K&R, retourne un pointeur sur «char» (sur «void» en C-ANSI) et nécessite une conversion de type (avec les risques d'oublis que l'on sait...).

```
float* ptr = new float;     // Réservation d'un
                            // réel

int* pti =new int[20];      // Réservation de 20
                            // entiers

Matrice *ptm =new Matrice; // Réservation
                            // d'une Matrice
```

La dernière déclaration est évidemment sous-jacente à la pré-définition d'une classe d'objets «Matrice».

En ce qui concerne la désallocation, préciser les tailles reste facultatif (bien que conseillé) :

```
delete ptr;
delete pti;      // Equivaut à delete [20] pti
```

L'allocation/désallocation par opérateurs devient spécialement intéressante en C++ lors de la mise en œuvre des

constructeurs et destructeurs avec surcharge des opérateurs; ce dont nous discuterons après avoir considéré le premier «plus» du langage.

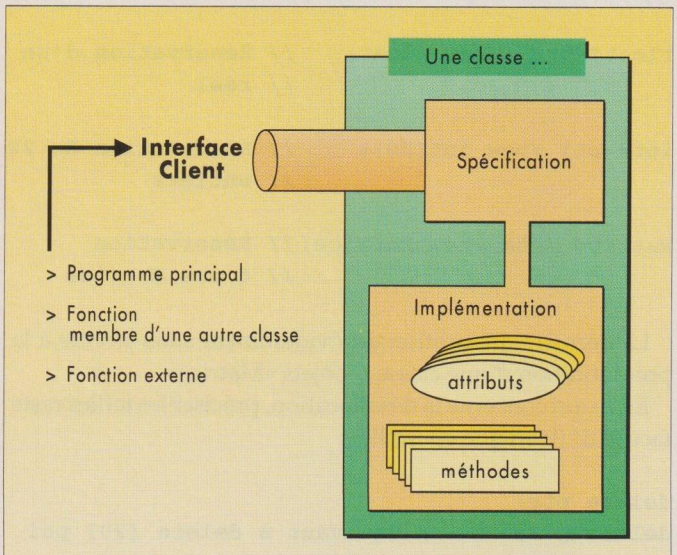
L'abstraction de données

En C, il n'existe pas implicitement de mécanismes d'abstraction (même si la notion de structure s'en rapproche). Sans toutefois égaler Ada (pour ses critères de modularité, de généricité et de compilation séparée), le C++ intègre un certain nombre de fonctionnalités autorisant une programmation à un niveau plus abstrait qu'en langage C : en particulier grâce aux principes de surcharge (overloading), à une pseudo-généricité, au mécanisme de constructeurs et à la possibilité de déclarer des classes d'objets.

Une classe constitue un type de données abstraites (voir Schéma 1) dont les attributs permettent de respecter deux principes fondamentaux de l'abstraction : l'encapsulation des données et le masquage de l'information (contrôle et protection des accès). On peut encore la considérer comme une extension du type «struct» dont certains champs peuvent être des fonctions (auxquelles on accède par les classiques opérateurs «.» et «-» selon le cas).

Considérons une classe Matrice : un programme «client» pourra déclarer :

```
main()
{
Matrice M1;      // principe d'encapsulation
//...
}
```



LE TYPE « CLASS » ABSTRACTION ET ENCAPSULATION

Pour illustrer le principe du masquage de l'information, envisageons la structure de la matrice (nombre de lignes, de colonnes, etc.). Cette dernière pourra être définie de manière privée («private») au sein de la classe même. Un programme désirant manipuler cette structure devra forcément passer par une fonction (méthode) publique de ladite classe. Le schéma 1 présente un exemple d'accès à une classe par un programme Client.

Le contrôle des accès sur les objets d'une classe s'exerce au moyen de quatre attributs : l'attribut «public» concerne les membres accessibles sans contrainte particulière; l'attribut «protected» protège les entités concernées des accès autres que ceux émanant des classes dérivées (principe de l'héritage); l'attribut «private» n'autorise l'accès qu'aux fonctions faisant partie de l'implémentation de la classe même; l'attribut «friend» s'utilise pour atteindre un membre privé d'une classe C1 à partir d'une classe de type C2.

Les constructeurs et les destructeurs

Le mécanisme de constructeurs et destructeurs permet de systématiser l'allocation et la désallocation mémoire pour les objets (instances) d'une classe. En termes de temps d'exécution, il remplace l'implémentation d'un «garbage collector» du type Smalltalk.

Un constructeur constitue une fonction membre d'une classe dont l'identificateur est celui de cette dernière. Une telle fonction se trouve implicitement invoquée par le compilateur au moment de la création d'une instance de la classe ou de l'utilisation de l'opérateur «new». Un constructeur surchargé peut entraîner à déclarer et/ou à initialiser un objet de différentes manières (il existe alors plusieurs constructeurs paramétrés différemment). Conceptuellement, un constructeur autorise également le contrôle le droit de création d'un objet, il peut être, en effet, privé, protégé ou public.

```
class Chaine
{
friend class Text;
public :

Chaine(char *);  // constructeur spécifique
Chaine();        // constructeur par défaut
~Chaine();       // destructeur private :
 Chaine( int ); // constructeur privé }
```

Dans l'exemple ci-dessus, les fonctions membres de la classe Text (classe «amie») ont la possibilité de déclarer une chaîne de 1024 octets comme suit :

```
Chaine Buff_edit(1024); //accès au
constructeur privé
```

Les autres entités ne peuvent effectuer que des déclarations du type :

```
Chaine s1;              // appel du constructeur
                        //  par défaut

Chaine s2=«bye»;        // équivaut à Chaine
                        // s2(«bye»);
```

En ce qui concerne la désallocation, il n'existe qu'un destructeur par classe, non paramétré et impossible à surcharger. Il est implicitement appelé dès qu'un objet devient invisible (fin d'un bloc, d'une fonction, etc.) ou pendant l'utilisation explicite de l'opérateur «delete».

En résumé, ce mécanisme offre deux avantages non négligeables au programmeur. D'une part, il peut particulariser (au niveau de la classe) la manière d'allouer la mémoire pour l'objet, d'autre part, le risque d'oublier une réservation ou une libération d'espace mémoire lors de la création ou de la destruction dynamique des objets est moindre qu'en C, puisque le compilateur invoque judicieusement le constructeur adéquat (...) et le destructeur.

Pseudo-généricité et overloading

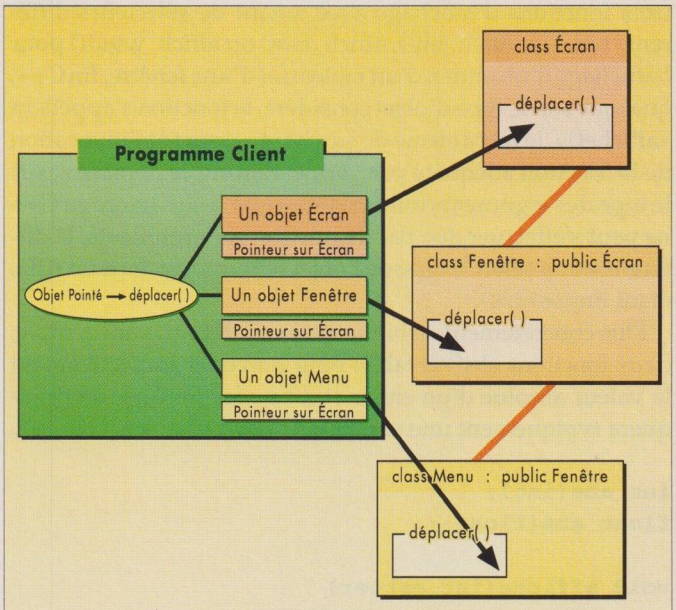
La généricité représente l'un des principes favorisant les possibilités d'abstraction d'un langage. Le C++ (version 2.0) n'autorise pas le paramétrage simple des types d'objets mais l'extension est prévue dans la version 2.1 du langage par l'attribut «template».

Sans être complètement générique (contrairement à Ada), avec la notion de pointeurs génériques (sur «void») et le type «reference», le langage dispose (pour des types de base uniquement) d'un embryon de généricité.

En fait, en C++, une variable du type «pointeur sur void» ou une fonction retournant ce dernier peuvent être assignées par un pointeur sur n'importe quel autre type de base (l'inverse restant impossible) :

```
void *ptr_divers;
int *ptr_ent = new int;
main()
{
ptr_divers = ptr_ent;   // Compilation
                        //correcte
ptr_ent = ptr_divers;   // Erreur de
                        // compilation
}
```

On peut ainsi implanter (pour les types de base du langage...) des fonctions génériques telles que la fonction d'échange suivante :



LE MÉCANISME DU POLYMORPHISME

```
void echange(void* &objet1, void* &objet2)
{
void *obj_temp = objet1;

objet1 = objet2;
objet2 = obj_temp;
}
```

Notons que les paramètres de la fonction sont de type «reference» (&), ce qui, conceptuellement, autorise bien la modification des paramètres en retour (donc l'échange effectif des objets, ici des pointeurs). Un programme principal pourra utiliser cette fonction de manière générique pour échanger des entiers, des réels, des caractères, etc,.à condition de manipuler des pointeurs :

```
main()
{
int *i = new int, *j = new int;
float *x = new float, *y = new float;
*i = 11;
*j = 22;
*x = 11.1;
*y = 22.2;

swap(i,j);      // Echange d'entiers...
swap(x,y);      // Echange de réels...
}
```

Le concept d'«overloading» (ou de surcharge) permet de ne pas tenir compte d'une fonctionnalité (surcharge fonctionnelle) ou d'une opération (surcharge opératoire). Par exemple, en langage C, il convient de définir respectivement

DU C VERS LE C++ : QUELS AVANTAGES RÉELS ?

trois fonctions d'affichage avec autant de sélecteurs différents tels que affich_int(), affich_tab() ou affich_wind() pour l'affichage d'un entier, d'un tableau et d'une fenêtre. En C++, quel que soit le type d'objet considéré, la fonction s'appellera «affiche()», le mécanisme de surcharge assurant l'invocation de la fonction adéquate par vérification sur le nombre et sur le type des arguments (on remarque que la surcharge en C++ ne peut s'effectuer que par distinction sur arguments, la différenciation selon le type de l'objet retourné, comme en Ada, étant impossible).

Plus concrètement, la bibliothèque C définit, par exemple, deux fonctions abs() et fabs() pour renvoyer respectivement la valeur absolue d'un entier et d'un réel. En C++, on pratiquera typiquement une surcharge :

```
int abs(int);
float abs(float);

void affiche(int entier)
{
printf(« %d »,entier);
};

void affiche(float reel)
{
printf(« %f »,reel); };

void affiche(Matrice mat)
{
// gestion de l'affichage d'une matrice...
};

main()
{
float x; int i;
Matrice m;      // surcharges fonctionnelles...
x=abs(3.14);
i = abs(-4);
affiche(i);
affiche(x);
affiche(m);
}
```

Selon le même concept, la surcharge des opérateurs (par le mot-clé «operator») autorise une notation plus intuitive : ainsi, il est plus naturel d'écrire MAT3 = MAT1 + MAT2 que M3=somme_matrice(M1, M2), pour additionner des objets de type Matrice ou encore : S3 = S1 + S2 pour concaténer des chaînes de caractères. Dans les deux cas, il y a, entre autres, surcharge de l'opérateur «+».

A titre d'exemple, le langage intègre une bibliothèque de classes «iostream.h» dans laquelle il y a surcharge des opérateurs de décalage «>» et «<» pour signifier plus intuitivement les opérations d'entrées et de sorties. Ceci évite au programmeur les ambiguïtés de formats %c, %s, %x, de

LES DIFFÉRENCES

Potentiellement compatibles, le C-ANSI et le C++ divergent néanmoins sur quelques points de détail qu'il est intéressant de préciser afin de mieux contrôler le portage d'un programme C en C++.
En premier lieu, tout programme C-ANSI faisant référence aux nouveaux mots-clés du C++ ne peut évidemment pas fonctionner sur tous les types d'ordinateurs. Ces mots réservés sont les suivants :
asm, class, delete, friend, inline, new, operator, overload, private, protected, public, template, this, virtual.
En C++, une déclaration telle que :

```
void fonction();
```

signifie que la fonction n'attend aucun argument; alors qu'en C-ANSI (pour raison de compatibilité avec le C-K&R), elle signifie que celle-ci escompte un nombre inconnu d'arguments de types quelconques. Notons que le prototypage valide aussi bien en C++ qu'en C-ANSI apparaît comme suit :

```
void fonction(void);
```

A une variable globale de type «const» est associée, par défaut en C++, la classe d'allocation «static». Alors qu'en C il s'agit de la classe «extern». Cette différence autorise en C++ l'utilisation de variables «const» dans des expressions constantes et permet, par exemple, de remplacer avantageusement la définition d'une macrofonction traitée classiquement par un préprocesseur en C :

```
const TAILLE = 100;
int tableau[TAILLE];   // Valide en C++
                       // Illégal en C-ANSI
```

En C++, la visibilité des éléments d'une énumération se limite à la structure dans de laquelle ils sont déclarés :

```
struct exemple
{
enum {droite,milieu,gauche};
// ...
};

int milieu;  // correct en C++,
             // Erreur en C-ANSI...
             // ...redéfinition d'identificateur
```

En C++, il n'est pas possible de sauter (par un «goto») au-dessus d'instructions d'initialisation (du fait que ces dernières font intervenir un constructeur). En C, cela reste possible, quoique potentiellement dangereux.

```
#include "chaine.h"
void fonc()
{
//...

goto bad_label;
{
int i=1;
Chaine str="danger";
```

C-ANSI / C++

```
bad_label :
// ...
}
}
```

En C++, le type d'un caractère littéral constant est «char» et non «int». Par exemple, sur une machine 16 bits (type «int» de quatre octets), on aura le comportement suivant :

```
taille=sizeof('?');    // taille=1 en C++
                       // taille=4 en C-ANSI
```

En C++, l'utilisation d'une fonction fct() non prototypée au préalable provoque une erreur. En C, le compilateur considère la déclaration implicite suivante «int fct()» sans vérification sur le type des éventuels paramètres :

```
main()
{
fct(1.41421);   // erreur en C++
                // int fct() par défaut
                // en C-ANSI
}
```

En C++, l'assignation d'un pointeur sur un objet de type void par un pointeur sur un objet d'un autre type nécessite un «cast» explicite. En C, ce dernier est implicite :

```
main()
{
int i=1;  int taille;
void *ptv=&i;          // Erreur en C++
                       // Correct en C-ANSI
char *ptc= ptv;        // Hum...
taille=strlen(ptc);    // ???  }
```

En C++, l'initialisation d'un tableau de N caractères par une chaîne constante de N caractères provoque une erreur :

```
char cartab[6]="azerty";
                  // Erreur en C++
                  // Correct en C-ANSI
```

En C++, un identificateur local de structure masque un quelconque identificateur global (d'une variable, d'une fonction, d'un type, etc.) :

```
int tb[TAILLE];
void locale()
{
struct tb { /* ... */ };
int size;
size=sizeof(tb);
            // En C++ taille de la structure
            // En C-ANSI taille du tableau
}
```

Pour obtenir la taille du tableau en C++, il faut utiliser l'opérateur de résolution de portée «::» comme suit :
sizeof(::tb);

pointeurs et d'objets pointés (*tab, tab) sur des fonctions aux résultats aussi imprévisibles que la météo (printf(), sprintf(), scanf(), sscanf()...). Les flots standard d'entrées, de sorties et d'erreurs étant respectivement repérés par «cin», «cout» et «cerr» :

```
main()
{
int val = 12;
float x;
Matrice m;
char *message = «Entrer un entier puis un
réel ? : »;
cout<<val;
cout<<m;          // surcharge à définir pour la
                  // classe Matrice
cout<< message << endl;
cin >> val >> x;
cout<<«Valeur de val = «<<val<<«x
                          = »<<x<<endl;
}
```

cout<<val est, en fait, interprété comme cout.operator<<(val), autrement dit l'appel à l'opérateur << de l'objet cout (instance de la classe ostream) avec, comme paramètre, l'entier val; «endl», défini tel un «manipulateur» du flot de sorties, permet d'envoyer une séquence «carriage return/line feed» et de vider le buffer associé au flot.

Pour information, l'une des surcharges de l'opérateur de décalage à gauche «<» dans «iostream.h» se définit comme suit :

```
ostream& ostream::operator<(ostream& flot,
int val)
{
// écriture de val dans le fichier associé à
«flot» return flot;
}
```

Le fait que la valeur retournée par l'opérateur soit une référence sur un objet «ostream» autorise les appels imbriqués du type :

```
cout < «valeur = » < val;
```

ce qui équivaut à :

```
(cout.operator<(«valeur =»)).operator<(val);
```

En tant que programmeur C averti, on imagine aisément la puissance et le confort des mécanismes tels que l'overloading, la généricité ou les constructeurs/destructeurs. Néanmoins, les quelques lignes de programmes qui précèdent tendent à montrer que dans la spécification même d'une classe (l'écriture des fonctions, la surcharge, le choix des attributs, etc.), les choses ne sont pas si triviales et qu'il faut déjà une solide expérience en langage C pour le codage. Il

importe de réaliser que le confort en C++ revient à celui qui manipule les classes (et non à celui qui les écrit) : «je déclare deux chaînes de caractères, je les concatène et j'affiche le tout», revient à écrire :

```
#include «chaine.h» //le plus difficile...
main()
{

Chaine C1 = «Langages »;
Chaine C2 = « Systèmes»;
Chaine C3;
C3 = C1+ «et» + C2;
affiche(C3);
}
```

Ainsi, en C++, il semble capital de disposer d'emblée d'une bibliothèque de classes bien fournie quitte, ensuite, à l'adapter selon le principe de l'héritage en programmation par objets.

Cela représente, en effet, le second «plus» par rapport au C. Car si toutes les fonctionnalités précédentes restent applicables dans un contexte de programmation procédurale classique, les notions d'héritage et de polymorphisme, indispensables dans un contexte de programmation par objets, devraient «interpeller» le programmeur C...

Le concept informatique d'héritage se rapproche de celui de l'héritage génétique où il y a similarité de comportement entre un «être» et un «ancêtre».

En C++, l'héritage se caractérise par la dérivation de classe. Une classe dérivée hérite des états (variables) et des comportements (fonctions) définis dans sa ou ses classes parentes. L'héritage peut être simple (une classe ne possède qu'une seule et unique classe parente) ou multiple (plusieurs classes parentes).

Considérons l'exemple suivant : deux classes «Cercle» et «Rectangle» dérivées d'une classe «Figure_fermée», elle-même provenant d'une classe «Forme». Une méthode Déplacer() décrite dans la classe Forme, restera probablement identique pour tout objet de la classe Figure_fermée et de ses dérivés. Par contre, il conviendra d'implanter une méthode Rotation() spécifique au niveau de la classe Cercle. Ainsi, le principe d'héritage évite de réimplanter ce qui n'est pas spécifique aux sous-classes et, par conséquent, de répondre à la réutilisation optimale des portions de codes déjà écrites !

Le polymorphisme

Nous avons déjà présenté deux cas de polymorphismes : l'opérationnel (l'overloading) et celui d'héritage (plus communément appelé «héritage»). Nous nous intéressons ici essentiellement au polymorphisme par inclusion, c'est-à-dire à la propriété d'un objet d'une classe C d'être considéré comme celui d'une autre classe K faisant partie de la même hiérarchie d'héritage que la classe C.

En C++, le polymorphisme par inclusion se distingue par l'implémentation de fonctions virtuelles («virtual»). Nous nous limiterons ici à une comparaison entre un style de programmation C et un autre de type C++ pour mettre en évidence l'intérêt de ce concept. Considérons la hiérarchie de classes suivantes : «Menu» dérivée de «Fenetre», elle-même dérivée d'«Ecran» et une méthode de déplacement d'un objet implémenté dans les trois classes (voir schéma 2). Un programme désirant activer un objet (sans savoir a priori s'il s'agit d'un Ecran, d'une Fenetre ou d'un Menu) s'articulera comme suit :

```
#include «Ecran.h»
main()
{

Ecran *ptr_objet;     // déclaration
                      // d'un pointeur
//...
// initialisation du pointeur selon
// les interactions de l'utilisateur
//...

ptr_obj-deplacer() // déplacement de l'objet
}
```

Si le langage enfreignait le polymorphisme, le message «ptr_obj-deplacer()» continuerait d'invoquer la méthode Déplacer() de la classe Ecran (classe d'appartenance du pointeur). Supporter le polymorphisme, c'est autoriser l'invocation implicite de la méthode associée à l'objet pointé (de type Ecran, Fenêtre ou Menu) sans connaître a priori ce dernier ! Pour assurer cela en C++, il conviendra d'implémenter la méthode déplacer() de la classe Ecran en tant que virtuelle. Dès lors, si l'on invoque des méthodes indirectement par des pointeurs sur Ecran (à l'aide de l'opérateur «-»), la classe d'appartenance de l'objet pointé, et non celle du pointeur, sera considérée pour sélectionner la méthode à appliquer.

Par rapport au C, ce principe de polymorphisme évite en C++ la lourdeur des tests imbriqués sur le type de l'objet pointé (avec des «if» ou des «switch case») pour décider (après conversion de type) de l'appel de la méthode adéquate :

```
// Dans un style «C»
#include «Ecran.h»
enum { Ecran, Fenetre, Menu };

void deplacer()
{
switch ( ptr_obj-typeOf() )
{
case Ecran      : ptr_obj-deplacer(); break;
```

```
case Fenetre    : ((Fenetre *)ptr_obj)-
                        deplacer();  break;
case Menu       : ((Menu *)ptr_obj)-deplacer();
break; default  : // gestion d'erreur...
}
// Dans un style «C++»
void deplacer()

{
ptr_obj-deplacer();
// Pas de tests sur le type de l'objet
// pointé
// la méthode invoquée sera toujours la
// bonne...
}
```

Ajoutons que, dans le cadre de cette hypothèse, une extension de la hiérarchie de classes (en créant, par exemple, une sous-classe «Menu_pop_up») ne modifie en rien le code de la méthode déplacer() déjà implémentée !

C ou C++, lequel choisir ?

Bien que séduisant au premier abord, le C++ recèle cependant de sérieux pièges (en plus de ceux déjà répertoriés en C) à savoir :

- les subtilités attachées à la manipulation des références;
- les différences notables entre affectation et initialisation;
- les appels en cascade des constructeurs et des destructeurs (pas toujours faciles à maîtriser pour peu que la classe considérée contienne un champ de type classe);
- les règles de visibilité lors de l'application des principes de programmation par objets;
- la manipulation des classes de base virtuelles en héritage multiple.

Pour le programmeur «Client» qui se contente d'instancier des classes existantes (ou dans le pire des cas de les spécialiser), le C++ est tout aussi agréable à lire qu'à écrire. Néanmoins, le programmeur amené à construire des classes d'objets devra, un jour ou l'autre, se confronter à l'écriture d'un code C des plus ésotériques avec des déclarations du style «int& (*obj[3])()» ou «int (&(*obj)())[3]» (à ne pas confondre, bien sûr).

Ces déclarations anecdotiques servent à rappeler que le C++ n'est pas un langage à mettre entre toutes les mains si l'on désire profiter pleinement de ses possibilités. Par ailleurs, la programmation dans un tel langage exige un réel effort d'analyse et de conception avant le codage effectif, sous peine de sombrer dans un imbroglio de classes et de sous-classes jusqu'au point de ne plus savoir où intervenir en cas de bogue.

L'expérience des sociétés ayant fait le pas du C vers le C++ tendrait à prouver la nécessité de mettre en œuvre des outils d'analyse et de conception orientés «objet», voire même de génération automatique de classes, étant entendu que la mise à disposition d'un débogueur «source» paraît in-dis-pen-sa-ble.

C++, le langage d'avenir

Déjà implanté en milieu professionnel sous Unix et VMS, le C++ se positionne sur le marché semi-professionnel et personnel sous MS-DOS et Mac-OS. Il existe actuellement une pléthore de produits C++ (au standard AT&T 2.0 pour les plus sérieux) intégrant des environnements de programmation de plus en plus complets (aides à la conception de classes, gestionnaires de bibliothèques, générateurs de classes, débogueurs multifenêtres...). Si un consensus général semble se dégager autour des produits de Borland et de Zortech sur PC les adeptes du Mac utilisent essentiellement Think C 4.0 (Symantec) et MPW C++ d'Apple (en version bêta 3.1).

Pour en revenir au domaine professionnel, les langages concurrents au C++ sont surtout Objective-C et Eiffel. Le premier, conçu par Brad Cox, a donné naissance à l'interface utilisateur de la machine NeXT (bien que C++ soit proposé sur cette machine). Le second se présente comme l'œuvre de Bertrand Meyer (président d'Interactive Software Engineering), dont la renommée n'est plus à faire en matière de génie logiciel.

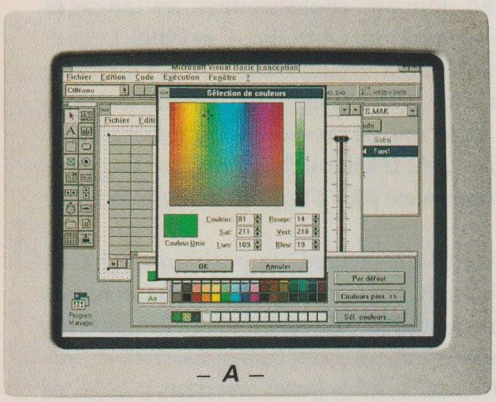
On reproche souvent à Objective-C sa position entre C et SmallTalk, compte tenu du fait que les programmeurs initiés aux deux langages restent plutôt rares. Eiffel fait l'unanimité dans le domaine universitaire et industriel mais, chez les utilisateurs, le poids d'AT&T semble peser fort dans la balance face à Interactive Software.

Parallèlement à ces langages compilés, SmallTalk tend à devenir une référence en matière de programmation par objets, mais il semble davantage voué au prototypage. Pour la réalisation d'applications, le C++ devrait, à terme, supplanter le langage C puisque, de toute façon, on peut l'utiliser sans faire obligatoirement de technique «objet». En effet, afin de faciliter le travail des programmeurs néophytes, certaines sociétés de développement logiciel utilisent déjà le C++ en programmation fonctionnelle à la place du C : l'avantage immédiat est probablement de disposer d'un langage de développement peu permissif. Mais les véritables gains en maintenance et en réutilisation passent par une application correcte des concepts «objet».

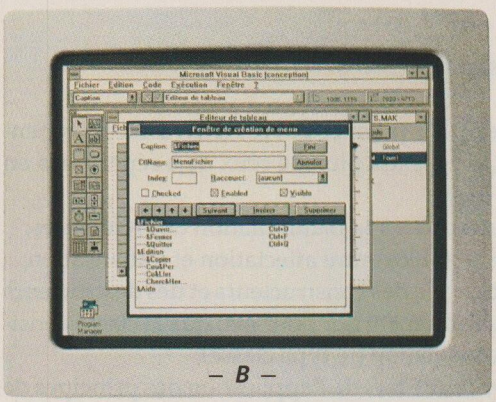
Le problème primordial reste donc la sensibilisation de toute une génération de programmeurs aux techniques d'abstraction et aux concepts de programmation par objets sans lesquels le langage de Bjarne Stroustrup proposerait les mêmes fonctionnalités que le langage C... quelques inconvénients en moins, toutefois ! □

Microsoft Visu Windows : il suffit pour faire de votr en programmatio

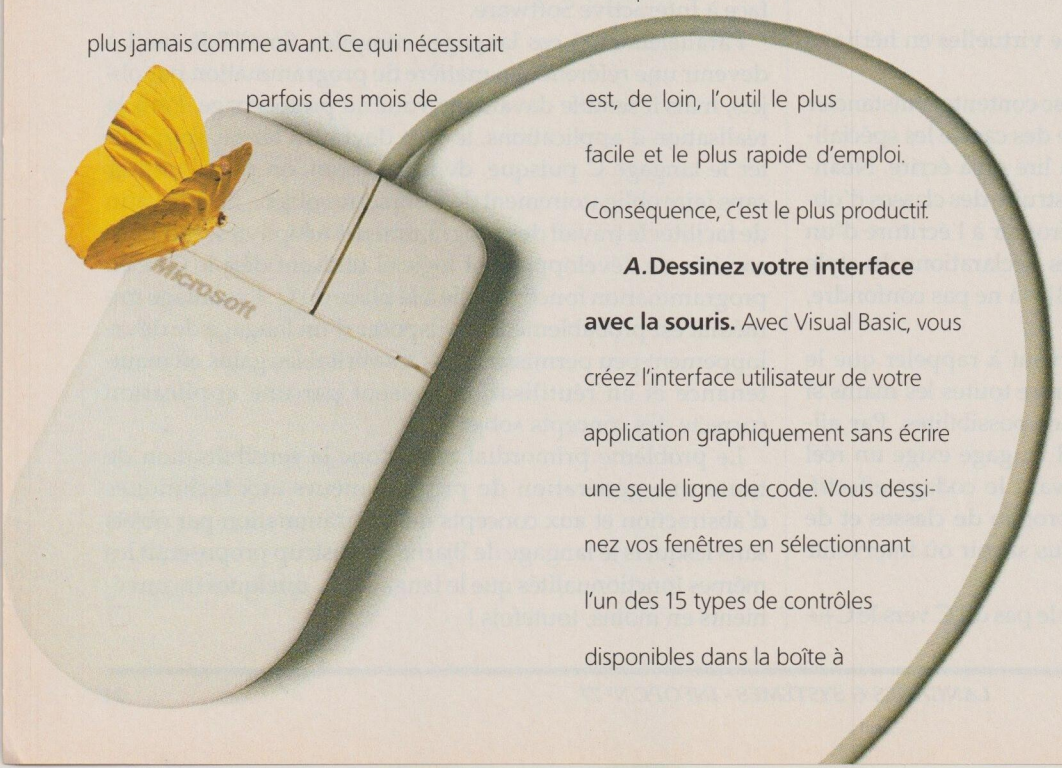
Avec Visual Basic 1.0, le monde de la programmation sous Windows ne sera plus jamais comme avant. Ce qui nécessitait parfois des mois de travail va maintenant se compter en journées. Que vous soyez développeur ou que vous souhaitiez vous lancer dans la programmation sous Windows pour vos besoins spécifiques, Microsoft Visual Basic pour Windows est, de loin, l'outil le plus facile et le plus rapide d'emploi. Conséquence, c'est le plus productif.



— A —



— B —



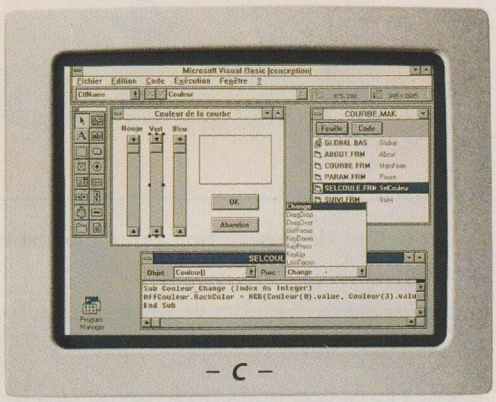
A. **Dessinez votre interface avec la souris.** Avec Visual Basic, vous créez l'interface utilisateur de votre application graphiquement sans écrire une seule ligne de code. Vous dessinez vos fenêtres en sélectionnant l'un des 15 types de contrôles disponibles dans la boîte à outils de Visual Basic. Pour les rendre plus attrayantes, vous pouvez y incorporer facilement des images que vous choisissez dans une bibliothèque de plus de 400 icônes fournie avec Visual Basic. Un éditeur d'icônes vous permet même de les modifier ou d'en créer de nouvelles.

B. **Créez vos menus visuellement.** Vous pouvez créer vos menus très facilement et associer chaque fonction à un accélérateur clavier grâce à la fenêtre de création de menus.

C. **Programmez à l'aide du langage le plus accessible de la micro-informatique.** La programmation est ensuite réservée aux traitements internes de vos applications. Elle s'effectue dans un

ıal Basic pour
de quelques clics
e micro un crack
n sous Windows.

langage Basic puissant et structuré, issu de QuickBasic et doté d'extensions orientées objets pour accéder facilement aux fonctionnalités de Windows. Vous saisissez votre code directement sous Windows en le rattachant très simplement aux éléments de votre interface graphique.

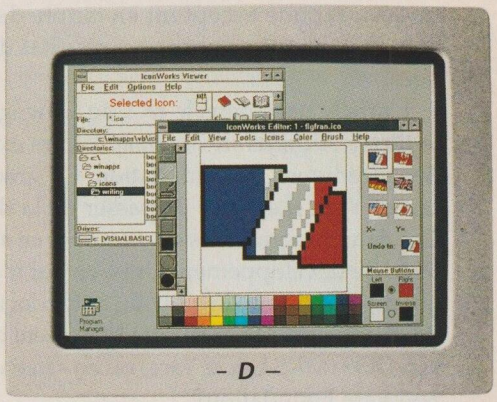


– C –

***D.* Exécutez et mettez au point votre application instantanément sous Windows.** La mise au point de vos programmes s'effectue directement sous Windows avec la possibilité d'exécuter le code pas à pas et de visualiser des variables.

Vous pouvez même modifier votre code pendant son exécution et relancer l'exécution au point d'arrêt ! Une fois la mise au point terminée, vous créez de véritables applications exécutables Windows qui sont diffusables sans run-time.

Accédez facilement à toute la puissance de Windows. Visual Basic est la plate-forme d'intégration idéale : la gestion du presse-papier, de l'impression, du lancement d'autres applications et l'échange dynamique de données (DDE) se programment très simplement. Vous pouvez également compléter votre application avec des bibliothèques dynamiques (DLLs) fournies par des éditeurs ou développées à l'aide de langages comme Microsoft C PDS 6.0, Microsoft FORTRAN PDS 5.1, Microsoft COBOL PDS 4.5 ou Microsoft Macro-Assembler PDS 6.0. Dans toutes les situations, vous pourrez construire votre solution. Microsoft Visual Basic est disponible en version française au prix de 1990 F HT.*



– D –

Pour toutes informations complémentaires, appelez le Service Clients au (1) 69 86 46 46 ou tapez 3616 code Microsoft.

Microsoft, 91957 Les Ulis cedex et à Infomart La Défense - Stand N° 276 niveau 2.

*soit 3 711,35 F TTC prix public conseillé au 1/09/91.

Microsoft®

LES LOGICIELS QUI DONNENT DES AILES.

Service lecteurs n° 6 (dernière page)

UNIX

LEX ET YACC

La pratique de la compilation

CLAUDE SIMON

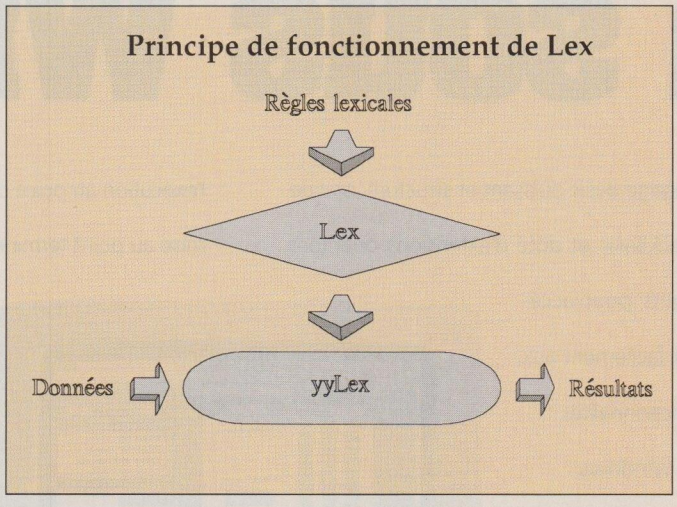
Si les techniques de compilation peuvent sembler ésotériques, dans la pratique leurs domaines d'applications sont nombreux. Outre l'écriture de compilateurs pour des langages informatiques, on peut les employer avantageusement pour l'écriture d'une calculatrice ou pour le traitement d'informations quelconques à partir du moment où elles ont une structure précise, même si celle-ci est complexe. La généralité de ce genre de programme devient manifeste dès lors que l'on considère le fait suivant : un compilateur est un programme acceptant en entrée des données structurées et donnant en sortie un résultat dépendant des données.

L'article paru dans le nº 1 de Langages & Systèmes sous le titre *L'Architecture des compilateurs* exposait les principes des techniques de compilation. Or grâce aux programmes Lex et Yacc, il est aisé de passer de la théorie à la pratique. Ces deux programmes sont fournis en standard avec le système de développement Unix, ce qui n'est pas une surprise quand on se rappelle qu'Unix a été conçu au départ pour des développeurs. On en trouve également des équivalents sous MS-DOS (MKS Lex & Yacc) ou en «free ware» (Bison de Free Software Foundation).

Lex

Lex et Yacc se complètent parfaitement. Ils ont été conçus pour fonctionner ensemble, même s'il est tout à fait possible de les utiliser séparément. Lex s'attend à ce qu'on lui donne un fichier contenant un ensemble de «motifs» à rechercher et d'actions associées à chacun de ces motifs. Il transforme ce fichier de description en un programme écrit en C capable de lire un fichier de données, d'exécuter les actions associées aux motifs qu'il a reconnus et d'écrire un fichier de résultats. Ce programme écrit en C contient une fonction principale qui s'appelle yylex(). Par abus de langage, ce programme généré par Lex sera parfois appelé «yylex()» dans la suite de cet article.

Le schéma suivant indique le principe de fonctionnement de Lex.



Un fichier de description pour Lex est composé de trois parties :

```
définitions
%%
motifs à reconnaître et actions associées
%%
sous-programmes de l'utilisateur
```

Chacune des parties est séparée de la suivante par une ligne comportant deux fois le caractère des pourcentages («%»). Seule la deuxième partie est obligatoire. Chacun des motifs à reconnaître est donné sous forme d'une expression régulière (*voir encadré*).

L'exemple L1 permet de créer un programme qui lit les caractères tapés au clavier et les écrit à l'écran en remplaçant les suites de caractères espace et tabulation par un seul espace. Dans cet exemple, «[\t]+» signifie le caractère espace ou tabulation répété une ou plusieurs fois de suite. L'action associée à cette règle suit immédiatement le motif à reconnaître. Cette action est composée d'une instruction écrite en C (ou plusieurs, si elles sont entourées par des accolades). Toujours dans l'exemple L1, l'action associée à l'unique motif à reconnaître est «printf(" ");» qui imprime un espace. Enfin, tous les caractères ne faisant partie d'aucun motif à reconnaî-

Les domaines d'application de la compilation sont nombreux. Son principe : les programmes acceptent en entrée des données structurées et élaborent en sortie un résultat dépendant des données. Avec Lex et Yacc, il est aisé de passer de la théorie à la pratique et de procéder, par exemple, à l'écriture d'une calculatrice.

tre sont imprimés sans modification par «yylex()». Il est difficile de faire plus simple !

Exemple L1

Pour créer le programme correspondant à ce fichier de description, il faut taper les commandes suivantes :

```
lex l1.l
mv lex.yy.c l1.c
cc l1.c -ll -o l1
```

Pour l'exécuter, il faut taper :

```
l1
```

Tout ce qui est tapé ensuite au clavier est répété à l'écran à chaque fois qu'on tape sur la touche <Return>. Pour sortir du programme il faut taper <Ctrl-D> (fin de fichier).

```
/* Nom du fichier: l1.l */
%%
[ \t]+          printf( " " );
```

L'exemple L2 correspond davantage à ce que l'on pourrait trouver dans un compilateur. Il sait reconnaître des identificateurs (par exemple des noms de variables ou de fonctions), des nombres entiers et des chaînes de caractères entre guillemets. Le listing est suffisamment explicite pour se passer de commentaires.

Exemple L2

Pour créer le programme correspondant à ce fichier de description il faut taper les commandes suivantes :

```
lex l2.l
mv lex.yy.c l2.c
cc l2.c -ll -o l2
```

Pour l'exécuter, il faut taper :

```
l2
```

Pour terminer le programme, il faut taper <Ctrl-D>.

Cet exemple lit l'entrée standard et remplace les identificateurs (suite de lettres chiffres et soulignés) par le mot «identificateur», les nombres entiers par le mot «entier», et les chaînes de caractères par les mots «chaîne de caractères». Il laisse intact tous les autres caractères.

```
/* Nom du fichier: l2.l */
%%
[A-Za-z_][A-Za-z0-9_]* {
    printf( " identificateur " );
  }
[0-9]+ {
    printf( " entier " );
  }
\"[^\"]*\" {
    printf( " chaîne de caractères " );
  }
```

EXPRESSIONS RÉGULIÈRES DANS LEX

x le caractère «x».
\x le caractère «x», même si x a une signification spéciale pour Lex.
[xy] le caractère «x» ou «y».
[x-z] n'importe quel caractère entre «x» et «z», bornes comprises.
[^x] n'importe quel caractère sauf «x».
. n'importe quel caractère sauf un saut de ligne ("\n").
^x le caractère «x» au début de la ligne.
x$ le caractère «x» en fin de ligne.
x? le caractère «x» optionnel.
x* le caractère «x» zéro, une ou plusieurs fois de suite.
x+ le caractère «x» une ou plusieurs fois.
x|y le caractère «x» ou «y».
x/y le caractère «x», mais seulement s'il est suivi par le caractère «y».
x{m,n} entre m et n fois de suite le caractère «x».
\"[^\"]*\" signifie donc le caractère «"» puis zéro, une ou plusieurs fois un caractère différent de «"» ([^\"]*) et le caractère «"» (\") une nouvelle fois.
^#.*$ signifie un caractère dièse en début de ligne (^#) puis zéro, un ou plusieurs caractères jusqu'à la fin de la ligne (.*$).

LEX ET YACC

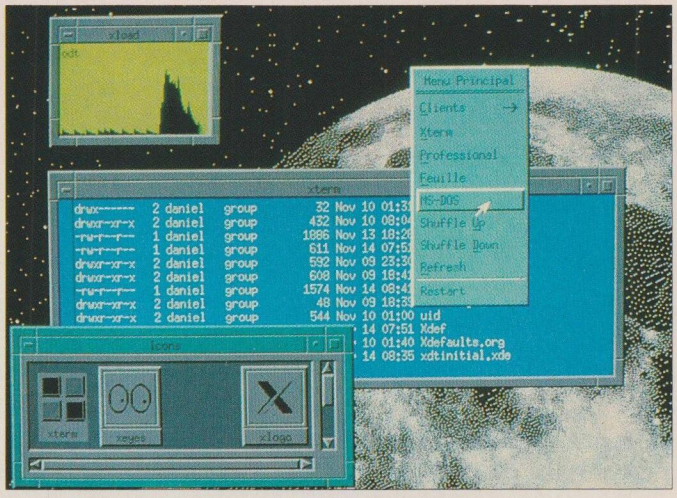
Lex offre de nombreuses autres possibilités, mais le fait que les programmes créés par lui ne gèrent que des caractères codés sur sept bits est rédhibitoire pour certaines applications (gestion des accents). Il sera donc parfois préférable d'écrire son propre analyseur lexical sans utiliser Lex. Toutefois, afin de ne pas sans cesse réinventer la roue et de gagner en productivité, il est conseillé de l'utiliser à chaque fois que cela sera possible.

Yacc

Yacc, quant à lui, est beaucoup plus puissant, et il sera particulièrement difficile de réécrire un équivalent de ce qu'il sait générer. De même que Lex, Yacc demande à ce qu'on lui donne un fichier contenant un ensemble de règles grammaticales et d'actions associées. Yacc transforme ce fichier de description en un programme écrit en C faisant appel à la fonction yylex() pour lire les caractères et exécutant les actions associées aux règles satisfaites. Ce programme contient une fonction principale qui s'appelle yyparse(). Par abus de langage, ce programme généré par Yacc sera parfois appelé «yyparse()» dans la suite de l'article. Le schéma ci-dessous indique le principe de fonctionnement de Lex et de Yacc lorsqu'ils sont utilisés conjointement. Un fichier de description pour Yacc se compose, lui aussi, de trois parties :

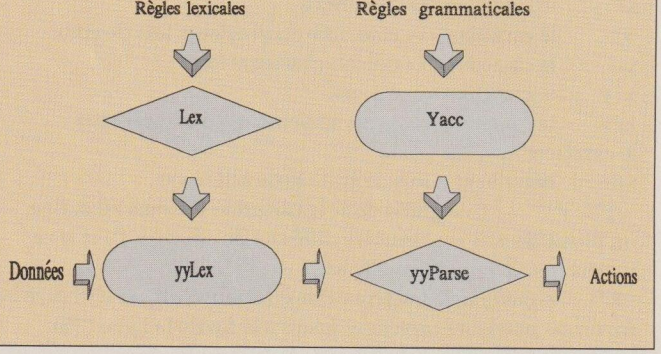
```
déclarations
%%
règles de grammaires et actions associées
%%
sous-programmes de l'utilisateur
```

Seule la deuxième partie est obligatoire. L'exemple composé de Y1 et de L3 analyse un fichier qui pourrait servir à paramétrer un programme. Dans le fichier L3 de description pour Lex, le but n'est plus d'afficher un texte modifié mais de préparer le travail pour l'analyseur grammatical. Les instructions «printf();» ont donc été remplacées par des instructions «return();». De plus, il n'est plus possible d'utiliser les actions par défaut du programme généré par Lex, car celles-ci consistent à afficher les caractères non reconnus par ailleurs. C'est pourquoi, dans cet exemple, figure la règle «[.\n] return(yytext[0]);» qui est destinée à reconnaître tout caractère n'ayant pas été reconnu par l'une des autres règles («.» représente n'importe quel caractère sauf le saut de ligne et «\n» représente le saut de ligne). «yytext» est une variable globale mise à jour par «yylex()». Elle contient les caractères ayant été reconnus par le motif dont on exécute l'action associée. «yylex()» peut retourner deux valeurs à «yyparse()». La première indique le mot-clé reconnu (COULEUR, IMPRIMANTE, chaîne, entier...). La deuxième permet de différencier le mot-clé reconnu. En effet, si, au niveau des règles de grammaire, le contenu d'une chaîne de caractères n'a pas d'importance, il faut bien, au niveau des actions associées aux règles, pouvoir retrouver ce contenu. La première valeur, obligatoire, est transmise par l'intermédiaire de l'instruction «return();». L'autre, facultative, est passée par l'intermédiaire de la variable globale yylval qui est définie par «yyparse()». Dans l'exemple donné, cette variable est utilisée lorsqu'un nombre ou une chaîne de caractères ont été reconnus. Cette variable étant de type entier, il faut le moyen de transformer un nombre ou une chaîne de caractères en entier. C'est l'objet des deux fonctions st_entier() et st_chaine(). Ces deux fonctions utilisent ici une méthode particulièrement inefficace mais très simple. Elles rangent les paramètres qui leur sont donnés dans des tableaux et retournent la valeur de l'indice correspondant dans le tableau. Dans un vrai compilateur, on évitera de ranger deux fois la même valeur, les chaînes de caractères seront gérées en longueur variable et, surtout, l'espace attribué au rangement de ces constantes sera alloué de manière dynamique en fonction des besoins. La ligne «^#.*$;» permet de ne rien faire lorsqu'une ligne du fichier de paramètres commence par un dièse («#»). Toute la ligne est ignorée et «yyparse()» ne sera jamais mis au courant du contenu de ces lignes. Ceci est d'au-



Un exemple d'utilisation de Lex et Yacc sous Unix.

Principes de fonctionnement de Lex et Yacc utilisés ensemble



tant plus intéressant qu'il serait difficile de donner une syntaxe correspondant à des commentaires ! Dans le même ordre d'idées, il existe deux motifs permettant de ne rien faire lorsque des espaces ou des tabulations sont rencontrées.

La première partie du fichier de description pour Yacc commence par définir les mots-clés (token en anglais) du langage à analyser. Comme il a déjà été dit, Yacc transforme ces mots-clés en «macros» et leur donne une valeur numérique automatiquement. Ces macros peuvent être utilisées entre autres par «yylex()» pour indiquer quels mots-clés ont été reconnus. La directive «%start liste_de_parametres» indique à Yacc que le programme qu'il va générer doit essayer de reconnaître cette règle. Si ce programme réussit, il retournera zéro, indiquant que tous les caractères lus étaient compatibles avec cette règle. Sinon, s'il y a incompatibilité entre les caractères lus et cette règle, il retournera la valeur un. La règle «liste_de_parametres» est définie de manière récursive. Cette règle est en fait composée de deux règles :
liste_de_parametres : parametre;
liste_d_parametres : liste_de_parametres 'n' parametre;
Ces deux règles de grammaire ayant la même partie gauche, elles peuvent être regroupées ensemble. On les sépare alors par le caractère « | ». La dernière est suivie du caractère point virgule («;»). On obtient alors :

```
liste_de_parametres :
    parametre
  | liste_de_parametres '\n' parametre
  ;
```

Cette règle de grammaire exprime que les paramètres doivent apparaître individuellement sur une ligne. La règle «parametre», quant à elle, est définie comme étant au choix : rien (pour pouvoir accepter les lignes vides et les lignes de commentaires), un paramètre de couleur ou un paramètre d'imprimante. On a dit précédemment que «yylex()» pouvait associer à chaque mot-clé une valeur grâce à la variable yylval. Ces valeurs sont mémorisées par «yyparse()» et lui servent à donner une valeur à chaque partie d'une règle. Dans le fichier de description pour Yacc, ces valeurs sont données par les pseudo-variables $$, $1, $2... $n. $$ correspond à la valeur de la partie gauche de la règle. Dans la règle «parametre: COULEUR type_de_couleur '=' couleur», $$ symbolise la valeur de la règle «parametre». $1 correspond à la valeur du premier élément de la partie droite de la règle. Dans ce même exemple, $1 est la valeur de COULEUR... $4 est la valeur du quatrième élément de la partie droite de la règle. Ici $4 est la valeur de «couleur».

Exemple Y1

Pour créer le programme correspondant à ce fichier de description, il faut taper les commandes suivantes :

```
lex l3.l
mv lex.yy.c l3.c
yacc -d y1.y
mv y.tab.c y1.c
mv y.tab.h y1.h
cc y1.c l3.c -ly -ll -o y1
```

Pour l'exécuter, il faut taper :

```
y1
```

Pour terminer le programme, il faut taper <Ctrl-D>.

```
/* Nom du fichier: y1.y */
%token  COULEUR, IMPRIMANTE
%token  BLEU, BLANC, ROUGE
%token  CARACTERE, FOND
%token  chaine, entier
%start  liste_de_parametres
%%
/*
** La règle suivante, qui est la règle
** principale, a deux composantes:
*/
liste_de_parametres :
    parametre
  | liste_de_parametres '\n' parametre
  ;
parametre :
    /* lignes vides et commentaires */
  | COULEUR type_de_couleur '=' couleur  {
      set_color( $2, $4 );
    }
  | IMPRIMANTE '=' chaine ',' chaine  {
      /*
      ** La première chaîne correspond à
      ** un ordre "spooler" ("lp" par
      ** exemple). La deuxième correspond
      ** à un fichier de description de
      ** l'imprimante.
      */
      set_imprimante( $3, $5 );
    }
  ;
couleur :
    BLEU  { $$ = 1;}
  | BLANC  { $$ = 2; }
  | ROUGE  { $$ = 3; }
  ;
type_de_couleur :
    CARACTERE  { $$ = 1; }
  | FOND  { $$ = 2; }
  ;
%%
/*
** Cette fonction est censé initialiser la
** couleur des caractères et du fond en
** fonction de ses paramètres.
*/
int  set_color( tdc, col )
int  tdc; /* type de couleur */
```

```
int  col; /* couleur */
{
    printf( "type de couleur %d = couleur
%d\n, tdc, col );
}
/*
** Cette fonction est censée initialiser le
** programme pour que les impressions
** ultérieures se fassent sur l'imprimante
** indiquée.
*/
int  set_imprimante( spool, impr )
int  spool;  /* ordre "spooler" */
int  impr;  /* nom de l'imprimante */
{
    printf( "spooler %d, imprimante %d\n",
spool, impr );
}
```

Exemple L3

La façon d'utiliser ce fichier est expliquée dans l'exemple Y1. Dans la première partie du fichier de description, tout ce qui se trouve entre les lignes «%{» et «%}» est recopié textuellement par Lex au début du programme généré.

Remarque : le fichier d'en-tête «y1.h» est la copie du fichier "y.tab.h" qui est généré automatiquement par Yacc lorsqu'on le lance avec l'option «d». Il contient des définitions de macros pour les mots-clés définis par les directives «%token».

```
/* Nom du fichier: 13.1 */
%{
# include  "y1.h
/* variable définie par Yacc */
extern int  yylval;
/* définition de macros */
# define  MAXCHAINE  10
# define  MAXENTIER  10
/*
** Variables globales servant dans les sous-
** programmes.
*/
char  Chaines[MAXCHAINE+1][80];
int  Entiers[MAXENTIER+1];
%}
%%
"COULEUR"  { return( COULEUR ); }
"IMPRIMANTE"  { return( IMPRIMANTE ); }
"BLEU"  { return( BLEU ); }
"BLANC"  { return( BLANC ); }
"ROUGE"  { return( ROUGE ); }
"CARACTERE"  { return( CARACTERE ); }
"FOND"  { return( FOND ); }
\"[^\"]*\"  {
    yylval = st_chaine( yytext );
    return( chaine );
  }
[0-9]+  {
    yylval = st_entier( yytext );
    return( entier );
  }
^#.*$  ;
" "  ;
\t  ;
[.\n] {
    /*
    ** Tous les autres caractères sont
    ** envoyés sans transformation à
    ** yyparse().
    */
    return( yytext[0] );
  }
%%
/*
** Transforme une chaîne de caractères en un
** entier permettant de la retrouver.
*/
int  st_chaine( str )
char  *str;  /* chaîne à ranger */
{
  /* indice pour les chaînes */
  static int  I_chaine;
  /* Dépassement de capacité du tableau? */
  if (I_chaine = MAXCHAINE)  return( 0 );
  ++I_chaine;
  strcpy( Chaines+I_chaine, str );
  return( I_chaine );
}
/*
** Transforme un nombre en un entier
** permettant de le retrouver.
*/
int  st_entier( str )
char  *str;  /* nombre à ranger */
{
  /* indice pour les entiers */
  static int  I_entier;
  /* Dépassement de capacité du tableau? */
  if (I_entier = MAXENTIER)  return( 0 );
  ++I_entier;
  Entiers[I_entier] = atoi( str );
  return( I_entier );
}
```

Avantages

Les avantages de Lex et Yacc apparaissent clairement sur cet exemple. Les règles sont indépendantes : il est possible de modifier la syntaxe du paramètre de couleur sans influencer

celle du paramètre d'imprimante. Quel que soit l'ordre dans lequel l'utilisateur donne ses paramètres dans son fichier de paramètres, ceux-ci seront reconnus correctement. Il est extrêmement aisé d'ajouter de nouvelles possibilités. Par exemple, pour reconnaître également la couleur «VERT», il suffit dans Y1 de rajouter une ligne «%token VERT» dans la première partie, une ligne « | VERT { $$ = 4; }» dans la règle «couleur» et dans L3 une ligne «"VERT" return (VERT);». Il n'y a rien d'autre à faire ! Un autre avantage de Yacc est la cohérence du langage décrit. En effet, si l'on rajoute un type de paramètre ayant besoin d'une couleur, on sera certain en utilisant la règle «couleur» que les mêmes couleurs seront gérées partout. Ceci évite bien des erreurs lorsque, plusieurs mois après avoir écrit un programme, on doit le modifier pour ajouter de nouvelles possibilités. En centralisant l'information, on limite ainsi les risques d'oublier un endroit à modifier.

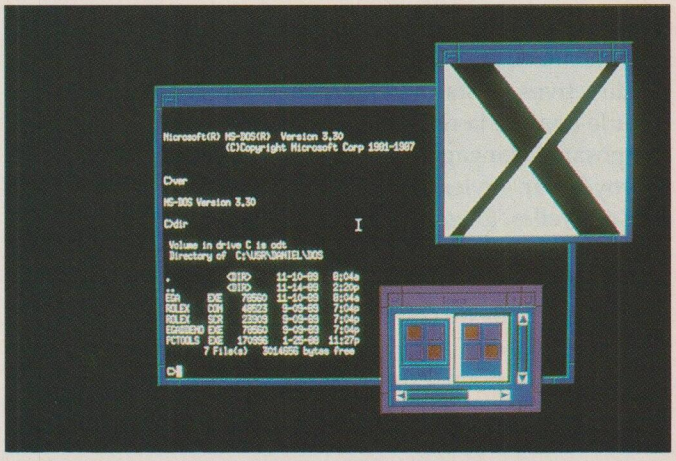
Les personnes perspicaces auront certainement remarqué qu'il est possible de transformer la règle «couleur» en un mot-clé pour obtenir exactement le même résultat. Il suffit pour cela de rajouter «%token couleur» dans la première partie de Y1, de supprimer la règle «couleur» et de faire les modifications suivantes dans L3 :

```
"BLEU"   { yylval=1; return( couleur ); }
"BLANC"  { yylval=2; return( couleur ); }
"ROUGE"  { yylval=3; return( couleur ); }
```

Les deux solutions sont strictement équivalentes du point de vue du résultat obtenu. L'on préférera toutefois cette deuxième solution dans le cas où la grammaire donnée à Yacc est complexe, de manière à simplifier le travail du programme généré par Yacc. En effet, toutes les «couleurs» sont équivalentes (au moins du point de vue grammatical!), et, puisque «yylex()» reconnaît chacune des couleurs et leur associe une action, il ne sert à rien que «yyparse()» fasse de même.

Réalisation d'une calculatrice

Jusqu'à présent, on s'est borné à donner des exemples simples mais concrets d'applications des techniques de compilation. L'exemple suivant dont on va discuter plus longuement est plus proche des compilateurs puisqu'il s'agit de la réalisation d'une calculatrice. En effet, Dans beaucoup de langages, la reconnaissance des expressions est au centre de l'analyse grammaticale. Cette calculatrice ressemble à celle que l'on trouve sous Unix. Elle permet de faire des calculs sur des nombres entiers dans n'importe quelle base comprise entre deux et trente-six (dix chiffres plus vingt-six lettres de «A» à «Z»), et elle possède vingt-six variables nommées de «a» à «z», permettant de mémoriser des résultats intermédiaires. Cette calculatrice permet également de faire très aisément des conversions entre deux bases différentes puisqu'il est possible d'indiquer que l'on désire saisir les nombres dans une base et les afficher dans une autre grâce aux pseudo-variables «ibase» et «obase».



Autre application de l'utilisation des programmes de compilation sous Unix

Dans l'exemple L4, on remarque que plusieurs motifs peuvent correspondre à une même séquence de caractères tapés par l'utilisateur. En effet, si l'utilisateur tape «ibase», il y a au moins trois possibilités :

```
«"ibase"  - >  return( base );»
«"i"  ->  return( variable );»  +  «"base"
 -> return( base );»
«"ibase"  -> return( variable );».
```

Le choix est déterminé par Lex. C'est le motif qui permet de reconnaître le plus de caractères simultanément qui est choisi. Ceci élimine le deuxième choix dans l'exemple précédent. Enfin, en cas d'égalité de longueur, c'est le motif qui apparaît le premier qui est choisi. Le troisième choix de l'exemple précédent est donc éliminé. Ainsi, dans un fichier de description pour Lex, il faut toujours commencer par les motifs correspondant à des cas particuliers (mots-clés, suite précise de caractères) et terminer par les motifs généraux (identificateurs, nombres, suite quelconque de caractères).

Comme on l'a déjà vu dans l'exemple Y1, tout ce qui se trouve entre les lignes «%XX» et «%YY» est recopié tel quel aussi bien par Lex que par Yacc. Cela permet, entre autres d'inclure des fichiers d'en-têtes et de définir des variables globales. Les variables de la calculatrice sont gérées de manière très simple : la variable C qui s'appelle «vars» est dimensionnée à vingt-six et chacun de ses éléments correspond à une variable de la calculatrice. Les noms des variables de la calculatrice tels que l'utilisateur les tape peuvent avoir n'importe quelle longueur, mais seul le premier caractère est pris en compte. Pour gérer un nombre quelconque de variables en tenant compte de plus de lettres, il suffirait de construire une table des symboles dans la règle de L4 qui reconnaît un nom de variable et d'associer à chaque variable

créée un numéro unique permettant de la retrouver pour lire son contenu ou pour la modifier. Dans Y2 on retrouve ensuite les directives «%start» et «%token» indiquant respectivement le nom de la règle principale et le nom des mots-clés composant le langage. Ces derniers sont susceptibles d'être renvoyés par «yylex()». Les directives suivantes («%left») sont nouvelles. Elles permettent d'accepter des grammaires ambiguës en indiquant à Yacc comment régler les conflits. L'expression «1+2*3» pourra selon les langages être évaluée comme «(1+2)*3» ou comme «1+(2*3)». Ces deux comportements peuvent être obtenus grâce à Yacc avec la même grammaire que celle donnée dans l'exemple Y2 en changeant juste l'ordre de priorité des opérateurs. Les opérateurs donnés dans une même directive «%left» sont considérés comme ayant la même priorité. Les directives sont données dans l'ordre croissant des priorités. Pour obtenir l'ordre d'évaluation «(1+2)*3» de l'expression «1+2*3», il suffit de mettre la ligne «%left '+' '-'» après la ligne «left '*' '/' '%'»! La directive «%left» indique également qu'en cas d'égalité de priorité, l'expression est évaluée de gauche à droite. Ainsi, l'expression «1-1-1» est évaluée comme «(1-1)-1». En APL, où l'ordre d'évaluation pour la même expression est «1-(1-1)» (!), il faudrait utiliser la directive «%right» pour indiquer qu'il faut évaluer de droite à gauche. On passe ensuite aux règles de grammaire. La règle «liste_de_commandes» n'appelle aucun commentaire. La règle «commande» décrit l'ensemble des commandes acceptées par la calculatrice. Pour rajouter une nouvelle commande, il suffit de la mettre à cet endroit. On pourrait, par exemple, rajouter une commande «Raz» qui réinitialiserait la calculatrice (passage en base dix et remise à zéro des variables). Il suffit pour cela d'ajouter une ligne «%token RAZ» dans la première partie de Y2, d'ajouter à la règle «commande» les lignes :

```
| RAZ {
  int i;
  ibase = 10;
  obase = 10;
  for ( i=0; i<26; i++ ) vars[i] = 0;
}
```

Il faut aussi rajouter une ligne «"Raz" return(RAZ);» dans L4. On remarque à la fin de cette règle «commande» l'emploi du mot «error» qui n'a été défini nulle part. Ce mot est réservé par Yacc pour la gestion d'erreurs. «error» permet d'accepter n'importe quoi. «error» peut être suivi d'un mot-clé. Dans ce cas, si une erreur se produit, «yyparse()» ignorera tous les caractères jusqu'à celui indiqué. Ceci permet de se «resynchroniser» et de se mettre dans un état connu. Dans l'exemple présent, on demande à «yyparse()» d'ignorer tous les caractères jusqu'au saut de ligne lorsqu'une erreur se produit. La ligne suivante tapée par l'utilisateur pourra donc être analysée correctement indépendamment des erreurs ayant pu survenir sur les lignes précédentes. «yyerrok» est une macro définie par «yyparse()». Elle permet de lui dire qu'à partir du moment où cette macro est utilisée, on n'est plus en état d'erreur. Sans cela, «yyparse()» considère que, pour sortir de cet état d'erreur, il lui faut reconnaître trois caractères valides consécutifs. L'appel à «yyerrok» est donc indispensable pour ne pas passer à côté d'erreurs successives. La gestion d'erreurs est souvent l'une des parties les plus difficiles à réaliser dans un compilateur, surtout si l'on désire pouvoir signaler plusieurs erreurs à la fois. Yacc propose un certain nombre de mécanismes simples qui permettent toutefois de gérer la plupart des cas. La règle suivante décrit la syntaxe acceptable pour une expression. Le seul point à noter à propos de cette règle concerne la façon ambiguë dont elle est décrite et la manière que Yacc a de régler les conflits grâce aux niveaux de priorités associés aux différents mots-clés. On a déjà expliqué plus haut comment indiquer ces niveaux de priorité grâce aux directives «%left» et «%right». Yacc attribue également un niveau de priorité à chaque règle. Ce niveau est déterminé par celui du dernier mot-clé de cette règle. Ainsi, les deux règles «expr: expr '-' expr;» et «expr: '-' expr;» ont le même niveau de priorité qui est celui associé à «'-'». Il est cependant possible de modifier ce niveau de priorité attribué par Yacc en utilisant la directive «%prec». C'est ce qui a été fait en écrivant «expr: '-' expr %prec MOINS_UNAIRE». Cette règle se trouve donc avoir la même priorité que MOINS_UNAIRE qui a été déclaré comme ayant la priorité la plus forte. On a vu dans l'exemple Y1 que «yyparse()» associe une valeur à chacune des règles qu'il reconnaît. Cette valeur permet ici de gérer les calculs. Toutefois, les calculs sont limités à des calculs sur des nombres entiers puisque cette valeur gérée par «yyparse()» est de type entier. Il est cependant possible de changer cela. Pour, par exemple, faire des calculs sur des nombres flottants double précision, il suffit d'ajouter entre les lignes «%XX» et «%YY» une ligne «typedef double YYSTYPE;». Le type YYSTYPE correspond au type des valeurs associées aux règles de grammaire. Il faut alors également modifier le type de la variable «vars» (double vars[26];), la fonction Affiche(), et le fichier de description pour Lex afin qu'il reconnaisse un nombre à virgule et qu'il mette sa valeur dans la variable yylval qui est en fait de type YYSTYPE. Tout le reste demeure inchangé. Il est relativement difficile de faire plus simple!

Exemple Y2

Pour créer le programme correspondant à ce fichier de description, il faut taper les commandes suivantes :

```
lex 14.l
mv lex.yy.c 14.c
yacc -d y2.y
mv y.tab.c y2.c
mv y.tab.h y2.h
cc y2.c l2.c -ly -ll -o y2
```

Pour l'exécuter, il faut taper :

```
y2
```

Pour terminer le programme, il faut taper <Ctrl-D>.

```
/* Nom du fichier: y2.y */
%{
    void  affiche();
    int  vars[26];
    int  ibase=10, obase=10;
%}
%start  liste_de_commandes
%token  nombre
%token  variable
%token  base
%left  '+' '-'
%left  '*' '/' '%'
%left  MOINS_UNAIRE
%%
liste_de_commandes :
    commande
  |liste_de_commandes commande
  ;
commande :
    '\n'
  | expr '\n'  { affiche( $1 ); }
  | variable '=' expr '\n'  {
      vars[$1] = $3;
    }
  | base '=' expr '\n'  {
        if (($3<2) || ($3>36))
        printf( "Base incorrecte (2<=
base<=36)\n" );
      else  {
        if ($1 & 1)  ibase = $3;
        if ($1 & 2)  obase = $3;
      }
    }
  | error '\n'  {
      printf( "Erreur de syntaxe\n" );
      yyerrok;
    }
  ;
expr :
    '(' expr ')'  { $$ = $2; }
  | expr '+' expr  { $$ = $1 + $3; }
  | expr '-' expr  { $$ = $1 - $3; }
  | expr '*' expr  { $$ = $1 * $3; }
  | expr '/' expr  { $$ = $1 / $3; }
  | expr '%' expr  { $$ = $1 % $3; }
  | '-' expr  %prec MOINS_UNAIRE  {
      $$ = - $2;
    }
  | variable  { $$ = vars[$1]; }
  |nombre
  ;
%%
void  affiche( nb )
int  nb;
{
  int  i, chiffre;
  char  tmp[33];
  if (nb < 0)  {
    printf( "-" );
    nb = -nb;
  }
  if (nb == 0)  {
    printf( "0\n" );
    return;
  }
  i = 0;
  do  {
    chiffre = nb % obase;
    if (chiffre  10)  tmp[i] = chiffre+'0';
    else    tmp[i] = chiffre+'A'-10;
    nb /= obase;
  } while (nb!=0);
  for ( ; i=0; i- - )
    printf( "%c", tmp[i] );
  printf( "\n" );
  return;
}
```

Exemple L4

La manière d'utiliser ce fichier est indiquée dans l'exemple Y2.

Remarque : le fichier d'en-tête «y2.h» est la copie du fichier «y.tab.h» qui est généré automatiquement par Yacc lorsqu'on le lance avec l'option «-d». Il contient des définitions de macros pour les mots-clés définis par les directives «%token».

```
/* Nom du fichier: 14.1 */
%{
# include  "y2.h"
/* variable définie par Yacc */
extern int  yylval;
/* variables globales */
int  i, chiffre;
%}
%%
"ibase"  { yylval = 1;  return( base ); }
"obase"  { yylval = 2;  return( base ); }
"base"  { yylval = 3;  return( base ); }
[a-z]+  {
      yylval = yytext[0] - 'a';
      return( variable );
    }
[0-9A-Z]+  {
      yylval = 0;
      for ( i=0; i(yytext); i++ ) {
      if ((yytext[i]='0') && (yytext[i]'9'))
        chiffre = yytext[i]-'0';
      else
        chiffre = yytext[i]-'A'+10;
      yylval = yylval*ibase + chiffre;
```

```
    }
    return( nombre );
  }
" "  ;
\t  ;
.  { return( yytext[0] ); }
\n  { return( yytext[0] ); }
```

Automates à états finis

Jusqu'à présent, il n'a été présenté que le moyen de décrire à Lex et Yacc un problème. Il est préférable de savoir comment fonctionnent les programmes «yylex()» et «yyparse()» pour pouvoir faire face à toutes les sortes de problèmes que l'on pourrait rencontrer. Bien évidemment, comme tout outil de développement qui se respecte, Lex et Yacc intègrent un mécanisme de déboguage qui permet de savoir ce qui se passe lorsque tout ne se déroule pas comme prévu. Ce mécanisme de déboguage est simple mais efficace. Il consiste à afficher les caractères lus et les actions entreprises par «yylex()» ou «yyparse()». Ces programmes «yylex()» et «yyparse()» sont des automates à états finis. L'automate commence son analyse dans l'état de départ. Dans cet état, ainsi que dans tous les autres, il possède une liste de caractères valides. Chacun de ces caractères valides permet de faire passer l'automate dans un autre état. Si un caractère non prévu arrive, c'est qu'il y a une erreur de syntaxe, et le mécanisme de gestion d'erreurs est déclenché. Il y a, de plus, un état final qui signale la fin possible de l'analyse. Dans cet état, la fin du fichier de données permet de terminer l'analyse. L'option «-v» de Yacc permet d'obtenir un descriptif de l'automate généré.

Etudions plus en détail le fonctionnement de l'automate sur la grammaire générée par la calculatrice. «yyparse()» gère deux piles : une pour savoir quels sont les états en attente et l'autre pour gérer les valeurs associées aux règles. L'automate connaît un nombre très limité d'opérations : L'opération «shift n» empile l'état n dans la pile d'états, elle considère que le caractère courant a été utilisé, et elle empile la valeur retournée par «yylex()» à propos de ce caractère dans la pile des valeurs. L'opération «reduce n» considère qu'on vient de reconnaître la règle numéro n. Elle effectue ensuite systématiquement l'action «$$ = $1;», puis éventuellement celle associée par l'utilisateur à cette règle. Cette action peut changer la valeur par défaut de la règle grâce à la pseudo-variable $$. Une fois ceci fait, «yyparse()» dépile dans les deux piles autant d'états et de valeurs qu'il y a d'éléments dans la partie droite de la règle de grammaire reconnue. Enfin, «yyparse()» empile la valeur de la règle reconnue. Cette opération ne modifie pas le caractère courant. L'opération «goto n» empile l'état n sans toucher à la pile des valeurs et sans modifier le caractère courant. L'opération «accept» termine l'analyse en signalant un succès. L'opération «error» termine également l'analyse, mais en signalant un échec.

Description de l'automate

```
state 0
  $accept : _liste_de_commandes $end
  error  shift 7
  nombre  shift 10
  variable  shift 5
  base  shift 6
  -  shift 9
  \n  shift 3
  (  shift 8
  .  error
  liste_de_commandes  goto 1
  commande  goto 2
  expr  goto 4
state 1
  $accept :  liste_de_commandes_$end
  liste_de_commandes:liste_de_commandes
commande
  $end  accept
  error  shift 7
  nombre  shift 10
  variable  shift 5
  base  shift 6
  -  shift 9
  \n  shift 3
  (  shift 8
  .  error
  commande  goto 11
  expr  goto 4
state 2
  liste_de_commandes :  commande_    (1)
  .  reduce 1
...
state 4
  commande :  expr_\n
  expr :  expr_+ expr
  expr :  expr_- expr
  expr :  expr_* expr
  expr :  expr_/ expr
  expr :  expr_% expr
  +  shift 13
  -  shift 14
  *  shift 15
  /  shift 16
  %  shift 17
  \n  shift 12
  .  error
...
state 10
  expr :  nombre_    (16)
  .  reduce 16
...
state 12
commande :  expr \n_    (4)
```

```
  .  reduce 4
state 13
  expr :  expr +_expr
  nombre  shift 10
  variable  shift 22
  -  shift 9
  (  shift 8      .  error
  expr  goto 24
...
state 15
  expr :  expr *_expr
  nombre  shift 10
  variable  shift 22
  -  shift 9
  (  shift 8
  .  error
  expr  goto 26
...
state 24
  expr :  expr_+ expr
  expr :  expr + expr_    (9)
  expr :  expr_- expr
  expr :  expr_* expr
  expr :  expr_/ expr
  expr :  expr_% expr
  *  shift 15
  /  shift 16
  %  shift 17
  .  reduce 9
...
state 26
  expr :  expr_+ expr
  expr :  expr_- expr
  expr :  expr_* expr
  expr :  expr * expr_    (11)
  expr :  expr_/ expr
  expr :  expr_% expr
  .  reduce 11
...
  15/3000 terminals, 3/1000 nonterminals
  17/2000 grammar rules, 34/5000 states
  0 shift/reduce, 0 reduce/reduce conflicts
reported
  17/1400 working sets used
  memory: states,etc. 254/40000, parser
13/70000
  7/600 distinct lookahead sets
  6 extra closures
  83 shift entries, 1 exceptions
  13 goto entries
  1 entries saved by goto default
  Optimizer space used: input 191/40000,
output 250/70000
  250 table entries, 196 zero
  maximum spread: 259, maximum offset: 61
```

Le tableau suivant donne l'exemple du fonctionnement de l'automate pour reconnaître l'expression «1+2*3». La première colonne indique quel est le caractère courant. La deuxième indique la composition de la pile d'états. La troisième indique la composition de la pile de valeurs et la dernière colonne indique l'opération effectuée par l'automate ainsi que la règle reconnue et son action associée dans le cas d'une opération «reduce». □

Car.	Etats	Valeurs	Opérations
'1'	0	0	shift 10
-	0,10	0,1	reduce 16 expr: nombre; {$$ = $1;}
-	0	0,1	goto 4
'+'	0,4	0,1	shift 13
'2'	0,4,13	0,1,'+'	shift 10
-	0,4,13,10	0,1,'+',2	reduce 16 expr: nombre; {$$ = $1;}
-	0,4,13	0,1,'+',2	goto 24
'*'	0,4,13,24	0,1,'+',2	shift 15
'3'	0,4,13,24,15	0,1,'+',2,'*'	shift 10
-	0,4,13,24,15,10	0,1,'+',2,'*',3	reduce 16 expr: nombre; {$$ = $1;}
-	0,4,13,24,15	0,1,'+',2,'*',3	goto 26
-	0,4,13,24,15,26	0,1,'+',2,'*',3	reduce 11 expr: expr *expr {$$ = $1 * $3;}
-	0,4,13	0,1,'+',6	goto 24
'\n'	0,4,13,24	0,1,'+',6	reduce 9 expr: expr + expr; {$$ = $1 + $3;}
'\n'	0	0,7	goto 4
'\n'	0,4	0,7	shift 12
-	0,4,12	0,7,'\n'	reduce 4 commande: expr \n; {affiche ($1);}
-	0	0,7	goto 2
-	0,2	0,7	reduce 1 liste de commandes:: commande {$$ = $1;}
-	0	0,7	goto 1
$end	0,1	0,7	accept

Bibliographie

*Fort des explications précédentes, chacun devrait être en mesure de passer à l'acte et d'utiliser sans trop de problèmes Lex et Yacc. La littérature sur ce sujet est assez pauvre, mais, pour aller plus loin, on pourra lire avantageusement le guide du programmeur d'Unix; Lex et Yacc y sont décrits dans deux chapitres particuliers. «*Lex & Yacc*» de Mason chez Addison Wesley est entièrement consacré à ces deux outils. Parmi les grands classiques sur les techniques de compilation, on peut citer «*Compilers: Principles, techniques and tools*» de Aho, Sethi et Ullman paru également chez Addison Wesley.*

LOGIN SCRIPT : L'autoexec réseau

EDOUARD TONNEAU

Le premier contact entre l'utilisateur et le réseau NetWare quel qu'il soit se fait par l'intermédiaire des «login scripts». Ceux-ci s'apparentent aux fichiers batchs du DOS, en gérant les accès au serveur. Ils sont exécutés lors de la connexion d'un utilisateur au serveur de fichiers et se subdivisent en trois types :

- celui par défaut (Default Login Script) : il s'accomplit lorsqu'aucun autre type de script n'est défini. Il s'agit du script utilisé lors de la première connexion du superviseur sur un nouveau serveur;
- celui du système (System Login Script) : il est commun à tous les utilisateurs, il contient les informations générales lors des connexions;
- le script utilisateur (User Login Script) : propre à chaque utilisateur, il contient les particularités d'environnement de chaque machine. Il est exécuté à la suite de celui du système, sous réserve de l'existence de ce dernier script.

Le macrolangage des login scripts est accessible par le biais de l'utilitaire Syscon, qui permet la gestion des utilisateurs du réseau. Syscon offre toutes les possibilités de création, effacement, modification ou configuration de chaque élément du réseau (utilisateur ou groupe), à condition de disposer des droits du superviseur. Un utilisateur peut seulement modifier son script de connexion. De surcroît, on doit respecter certaines règles dans l'emploi des login scripts :

- les majuscules et les minuscules restent indifférenciées, sauf pour les variables qui sont toujours saisies en caractères majuscules;
- la ligne de commande ne peut contenir plus de 150 caractères;
- chaque ligne de script ne doit comporter qu'une seule commande;
- une remarque peut être placée derrière n'importe quelle commande, excepté Attach, Comspec, DOS SET, Exit, Map et Write.

Le Login Script System est commun à tous les utilisateurs. Il ne peut être modifié que par l'administrateur de réseau.

Les commandes

ATTACH

permet de s'attacher à un autre serveur, tout en restant «loggué» au serveur actuel.

Syntaxe :
ATTACH <nom_de_serveur>
</nom_d'utilisateur><;mot_de_passe>
Exemple : ATTACH serveur1/guest;alain

```
NetWare System Configuration  V2.20d        Tuesday  15 October 1991  16:49
                              User ITI On File Server TECH

                        System Login Script

BREAK OFF
DOS BREAK OFF

WRITE " "

MAP DISPLAY OFF

MAP INS S1:=sys:public
MAP S16:=sys:login
MAP S16:=vol1:ccdata

MAP W:=vol1:textware
MAP INS S4:=W:
DOS SET textware="W:\\textware"
#SYSTIME

#CLR
```

Suivant en cela les systèmes d'exploitation DOS, OS/2 et Unix, NetWare permet de créer des fichiers de traitement par lots. Bien qu'explicitées dans la documentation, ces commandes demeurent sous-exploitées par suite d'une présentation austère et dépourvue d'exemples pratiques. Une mise au point s'avérait donc nécessaire.

BREAK ON / OFF

avec l'option Off désactive l'action des touches CTRL-C ou CTRL-Break lors du déroulement d'un login script. Cette commande est très utile pour interdire l'arrêt d'un login script par un utilisateur.

Syntaxe : BREAK ON ou BREAK OFF

COMSPEC

spécifie le chemin d'accès au fichier Command.COM, pour pouvoir recharger l'interpréteur de commandes.

Syntaxe :
COMSPEC = <drive:></nom_de_fichier>
Exemple :
COMSPEC = F:/command.com

DISPLAY

affiche le contenu d'un fichier texte au format ASCII, sans tenir compte des codes Escape ni des caractères de contrôle inclus dans celui-ci.

Syntaxe : DISPLAY <volume>
<répertoire/> <nom_de_fichier>
Exemple :
DISPLAY sys:public/info.txt

DOS BREAK ON / OFF

avec l'option OFF désactive l'action sur les touches CTRL-Break dans les programmes ou sous DOS, après déroulement du login script. Attention à ne pas confondre cette commande avec Break Off.

Syntaxe :
DOS BREAK ON ou DOS BREAK OFF

DOS SET

permet l'emploi des variables d'environnement DOS, tout comme l'ordre SET sous DOS.

Syntaxe :
DOS SET <variable> = <valeur>
Exemple : DOS SET PROMPT = «PP»

DOS VERIFY ON / OFF

lorsqu'il est forcé à ON, active la vérification de toute copie effectuée sur un drive local. Cette commande ne se voit prise en compte que si l'ordre Copy est lancé avec l'extension /V.

Syntaxe :
DOS VERIFY ON ou DOS VERIFY OFF

DRIVE

autorise la spécification du drive utilisé par défaut.

Syntaxe : DRIVEé:

EXIT

permet de quitter un login script avec, éventuellement, le lancement d'un programme.

Syntaxe :
EXIT ou EXIT <nom_de_fichier>

FDISPLAY

identique à Display, offre l'affichage d'un fichier textes au format Ascii, mais

```
NetWare System Configuration  V2.20d          Tuesday  15 October 1991  16:56
                              User ITI On File Server TECH

                    Login Script For User COMPTA

    MAP ERRORS OFF

;Informations relatives à la connexion

    WRITE "Bonjour %FULL_NAME"
    WRITE "Vous etes le %STATION e connecte"
    WRITE "Votre No. ETHERNET est le %P_STATION"

;Attachement à un autre serveur

    ATTACH srv_lyon/henri

;Gestion des différents drives

    MAP S:=srv_lyon/sys:
    MAP S16:=sys:system
    MAP G:=vol1:
```

Exemple de User Login Script propre à chaque utilisateur connecté au réseau. Il peut être modifié aussi bien par le superviseur que par l'utilisateur.

LOGIN SCRIPT : L'AUTOEXEC RÉSEAU

en gérant les caractères de contrôle et les codes Escape.

Syntaxe :
FDISPLAY <volume:> <répertoire/> <nom_de_fichier>
Exemple :
FDISPLAY sys:travail/textes/info.txt

FIRE PHASERS

génère un petit bip sur le haut-parleur de la station.

Syntaxe : FIRE PHASERS x TIMES, avec x représentant le nombre de bips, allant de 1 à 8.

GOTO

permet d'exécuter une partie d'un login script à la suite d'un test conditionnel. Suivant en cela le Basic et les fichiers Batch, cette commande effectue un saut vers un label prédéfini.

Syntaxe : GOTO <label>
Exemple :

```
IF DAY_OF_WEEK = «sunday»
GOTO deconnect
EXIT «menu main.mnu»
deconnect:
EXIT «logout»
```

IF & THEN

Ces commandes testent une condition par vérification de l'égalité, de l'inégalité, de l'invalidité par comparaisons d'infériorité ou de supériorité, au sein d'un login script.

Les tests d'égalité ou d'inégalité sont les suivants :

ÉGALITÉ	INÉGALITÉ
... is ...	... is not ...
=	!=
==	<> ou #
... equals ...	... does not equal ... ou ... not equal to ...

Les tests d'infériorité ou de supériorité sont :

> ou	... is greater than ...
< ou	... is less than ...
>= ou	... is greater than or equal to ...
<= ou	... is less than or equal to ...

Le terme Value déclare un test sur une valeur numérique, tandis que les autres portent par défaut sur des valeurs Ascii. Les variables utilisées lors de ces tests conditionnels se trouvent décrites plus loin.

Syntaxe :
IF <condition> THEN <commande>
Exemple :

```
IF LOGIN_NAME <> «supervisor»
THEN EXIT «logout»
IF VALUE NDAY_OF_WEEK = «1»
THEN EXIT «logout»
```

INCLUDE

lance un login script à l'intérieur d'un autre. Le fichier émis par cette commande doit être au format texte et se composer de commandes login script standard.

Syntaxe : INCLUDE
<volume:><répertoire\><nom_de_fichier>
Exemple :
INCLUDE sys:public\info.msg

EXEMPLE DE LOGIN SCRIPTS POUR L'UTILISATEUR

```
; LOGIN SCRIPT
; USER
; ITI
; 1989/90/91
+-----------------------------------------------+
; suppression de l'écho
        MAP DISPLAY OFF
; pas de messages d'erreur
        MAP ERRORS OFF
; informations relatives à la connexion
        WRITE «Bonjour %FULL_NAME»
        WRITE «Vous etes le %STATION e connecte»
        WRITE «Votre No. ETHERNET est le %P_STATION»
; attachement à un autre serveur
         ATTACH srv_lyon/henri
; gestion des différents drives
        MAP S:=srv_lyon/sys:
        MAP S16:=sys:system
        MAP G:=vol1:
        MAP H:=vol2:%LOGIN_NAME
        MAP I:=sys:utils\lanassis
        MAP M:=vol1:pcmdata
; redirection des impressions sur le serveur
  #capture to q=printq_0 nb nff ti=2
; changement de drive
        DRIVE H:
; sortie vers un menu
        EXIT «menu gestion.mnu»
; fin du script+
```

MACHINE NAME

indique le nom de la machine utilisée en tant que station. Celui spécifié est limité à 15 caractères.

Syntaxe : MACHINE NAME = «nom»
MACHINE = «nom»
Exemple :
MACHINE = «COMPAQ 386/25e»

MAP

similaire à l'ordre MAP, lancé sur la ligne de commande, elle permet d'attribuer un drive logique à un répertoire particulier du réseau. Les options sont identiques à celles de la commande MAP.

Syntaxe :
MAP, utilisé seul, affiche les drives logiques assignés pour cette station.
MAP <drive:> = <volume:><répertoire>

relie le drive spécifié au chemin.
MAP <drive:> = <drive:>
assigne le second drive au chemin du premier.
MAP INSERT <search drive:>= <volume:><répertoire>
attache un chemin d'accès ainsi qu'un drive au répertoire spécifié.
MAP ROOT
remplace le répertoire principal par celui dans lequel MAP ROOT est exécuté.
MAP DISPLAY ON
équivaut à un «echo on» sous DOS.
MAP DISPLAY OFF
remplace le «echo off» sous DOS.
MAP ERRORS ON
active l'affichage des messages d'erreur.
MAP ERRORS OFF
désactive l'affichage des messages d'erreur.
Exemple : MAP G:=vol1:applis\base

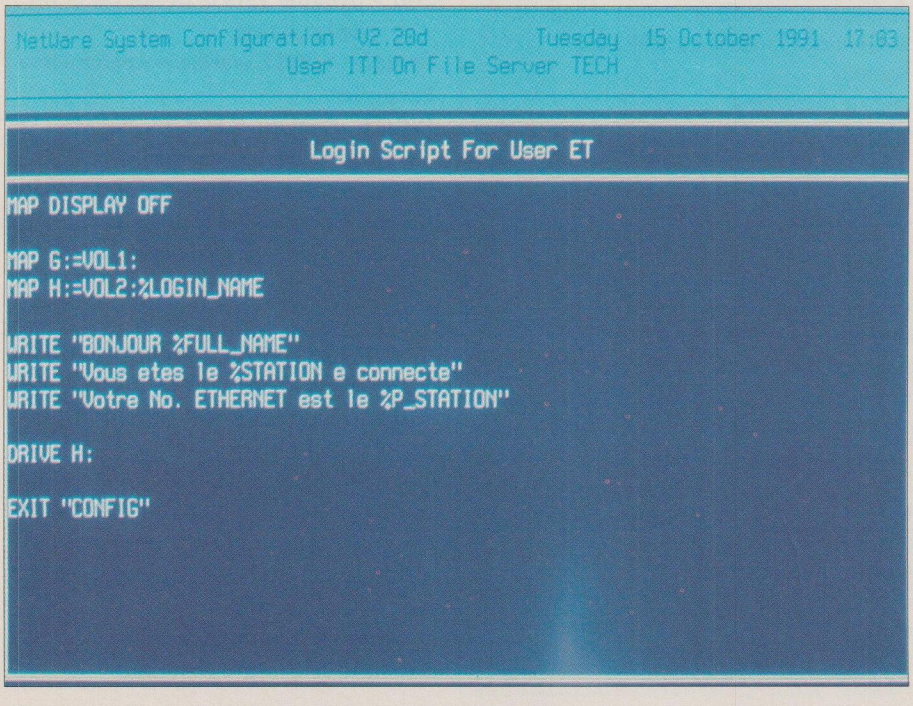
PASSWORD_EXPIRES

utilisé dans une ligne de tests, avec les commandes If Then étudiées plus haut, il autorise la vérification de l'état du mot de passe.
Syntaxe : IF PASSWORD_EXPIRES = <nombre_de_jours_avant_expiration> THEN ...
Exemple : IF PASSWORD_EXPIRES = «1» THEN WRITE «dernier jour de

EXEMPLES DE LOGIN SCRIPTS POUR L'ADMINISTRATEUR DE RÉSEAU

```
; LOGIN SCRIPT
; SYSTEME
; ITI
; 1989/90/91
; +-----------------------------------------+
; interdit le CTRL-C et le CTRL-Break
        BREAK OFF
        DOS BREAK OFF
; echo off
        MAP DISPLAY OFF
; pas de messages d'erreur
        MAP ERRORS OFF
; gestion des différents drives, communs à tous les utilisateurs
        MAP INS S1:=sys:public
        MAP INS S2:=sys:login
        MAP O:=vol1:applis
;redirection des impressions sur le serveur, dans une queue d'impression par défaut
        #capture j=default
;changement de drive
        DRIVE H:
;fin du script
```

Toutes les informations relatives aux login scripts, à Syscon et, par principe, à tous les outils fournis avec NetWare sont accessibles sur le serveur NetWare par l'utilitaire Folio (ce dernier accompagne avec chaque kit réseau) et dans le manuel Quick Access Guide.
Vous devez maintenant être capable de modifier votre propre login script sur un réseau NetWare, de la version 2.2 à la version 3.11. N'hésitez pas à faire parvenir vos découvertes et vos astuces sur les scripts de connexion, nous publierons les meilleures.



L'utilitaire SYSCON.EXE affiche des informations relatives à la connexion de l'utilisateur (identité de ce dernier, heure, date, et version du programme).

LOGIN SCRIPT : L'AUTOEXEC RÉSEAU

LES VARIABLES

Les variables, utilisables aussi bien par les tests If Then que par la commande Write, nécessitent obligatoirement la saisie en caractères majuscules. Dans la liste qui suit, les dates et heures sont données en fonction du temps serveur, utilisé comme référence. Il faut également retenir que les commandes, les remarques et textes ne supportent pas les caractères accentués.

Variable	Contenu de la variable
HOUR	heures du serveur,
valeur : 1 à 12	
HOUR24	heures du serveur,
valeur : 00 à 23	
MINUTE	minutes du serveur,
valeur : 00 à 59	
SECOND	secondes du serveur,
valeur : 00 à 59	
AM_PM	avant midi ou après midi,
valeur : am ou pm	
DAY	jour du serveur,
valeur : 01 à 31 N	
NDAY_OF_WEEK	nº. du jour du serveur,
valeur : 1 à 7, avec 1 pour sunday	
DAY_OF_WEEK	nom du jour du serveur,
valeur : sunday ...	
MONTH	mois du serveur,
valeur : 01 à 12	
MONTH_NAME	nom du mois du serveur,
valeur : january ...	
YEAR	année du serveur,
valeur : 1990, 1991 ...	
SHORT_YEAR	année réduite du serveur,
valeur : 90, 91 ...	
LOGIN_NAME	login name de l'utilisateur
FULL_NAME	nom complet de l'utilisateur, saisi sous Syscon
STATION	nº. de la station, par ordre d'arrivée sur le réseau
P_STATION	adresse physique de la station, format : 12 digits
USER_ID	identificateur correspondant à l'utilisateur
FILE_SERVER	nom du serveur
NETWORK_ADDRESS	adresse interne du serveur
GREETING_TIME	moment de la journée,
valeur : morning, afternoon ou evening	
OS	système d'exploitation de la station
OS_VERSION	version du système d'exploitation de la station
SHELL_TYPE	version du shell (système de connexion) utilisée
MACHINE	type de la machine, indiqué dans le fichier de configuration du shell
SMACHINE	type résumé de la machine, issu du fichier de configuration du shell (nommé «Shell.CFG»)
ERROR_LEVEL	nº. d'erreur, gérable de la même manière qu'en fichier batch,
valeur : 0 en l'absence d'erreur, autre valeur d'apparition de fautes.	
MEMBER_(OF)_	groupe teste l'appartenance au groupe spécifié

Exemples : WRITE «Good »; GREETING_TIME; «, »; LOGIN_NAME
IF SHELL_TYPE DOES NOT EQUAL «V5.00» THEN EXIT «logout»
WRITE «Vous êtes le %STATION e connecte.» \n

Toutes ces commandes et variables se trouvent rappelées, en résumé, dans l'aide incorporée dans Syscon. Cette aide en ligne est accessible par la touche F1.

travail. Renouvelez vite votre mot de passe.»

PAUSE

génère une pause dans le déroulement du login script, comme l'ordre Pause dans un fichier batch.

Syntaxe : PAUSE

REMARK

laisse une remarque dans le contenu du login script.

Syntaxe : REMARK ou REM ou ;<texte>

WRITE

affiche une ligne de message. Suivie de l'extension \r, Write ajoute à la ligne de texte un code «retour chariot». Si l'extension est \n, la ligne de caractères se trouve suivie du code «à la ligne», et si l'extension est \7, Write fait suivre le texte d'un bip. Write accepte les mêmes variables que les tests If Then, mais reconnaît aussi différents codes, permettant des opérations au sein de la ligne de texte suivant Write. Ces codes comprennent :

;	permet une concaténation,
*	effectue une multiplication,
/	exécute une division,
%	retourne le modulo,
+	additionne,
-	soustrait,
< ou >	tronque vers le début ou la fin de la chaîne.

Syntaxe : WRITE «<texte>» <paramètre>
Exemple :
WRITE «Bonjour, %LOGIN_NAME» \n

A noter que les variables utilisées dans une chaîne de texte avec Write doivent être précédées du symbole % et toujours saisies en majuscules.

#

autorise l'exécution d'ordres externes à l'interpréteur de commandes login script. Ce symbole est très utile pour lancer des fichiers .COM ou .EXE au sein d'un login script. Rappelons que les fichiers .BAT ne peuvent être lancés par cette option car l'interpréteur de commandes batchs entre en conflit avec le login script.

Syntaxe : # <commande_externe>
Exemple : # capture j=default □

MESSAGE AUX DEVELOPPEURS

Chauffez vos applications au rouge.

En profitant du "Professional Developer's Program" de Novell, vous forgez des logiciels gagnants pour vos applications travaillant en réseau.

Vous développez pour la plus grande base installée : 1 million de sites et 10 millions d'utilisateurs dans le monde travaillent sous NetWare.

Vous apportez à ces utilisateurs les performances de sécurité et de transparence de NetWare, y compris en architecture client-serveur.

Vous développez dans tous les environnements, DOS, OS/2, Windows, Macintosh, Unix...

Vous utilisez des outils de développement exceptionnels pour un prix exceptionnel.

Vous bénéficiez d'un programme d'information et de formation réservé aux seuls "Novell Professional Developers".

Informez-vous en renvoyant le coupon-réponse.

Le passé, le présent, l'avenir de l'informatique réseau

Je voudrais être "Novell Professional Developer"

☐ Informez-moi sur le "Professional Developer's Program" de Novell.
☐ Informez-moi sur les produits de développement Novell.

Nom _____
Société _____
Adresse _____
_____ Téléphone _____

Novell-Tour Anjou - 33 Quai de Dion Bouton,
92814 PUTEAUX Cedex-Tél. : 47 75 09 09-Fax : 47 78 94 72

PARD communication - Service lecteurs n° 7 (dernière page)

SYSTÈME

VESA : MODE D'EMPLOI

YVES GUITTARD

Ce document décrit les mécanismes des modes et fonctions de l'interface standard VGA étendue tels qu'ils ont été approuvés par le comité principal du Vesa. Ils devront également être utilisés pour les modes qui pourraient être ajoutés à un VGA non standard, afin que les développeurs puissent les utiliser sans avoir à entrer dans les détails complexes du matériel VGA cor-respondant. La spécification donne les définitions des modes VGA étendus ainsi que des fonctions à utiliser dans les logiciels d'application, nécessaires à la compréhension des caractéristiques de ce mode vidéo ainsi qu'à la manipulation des mémoires étendues associées à ces modes.

L'adaptateur VGA (Video Graphics Array) d'IBM est devenu le standard de fait sur PC dans le domaine de l'affichage couleur. Nombreuses sont les cartes compatibles présentes sur le marché et disposant d'un Bios, de modes graphiques étendus, voire de facilités de traitement graphique. Une concurrence acharnée a fortement amélioré le rapport performance/prix de ces produits, pour le plus grand bénéfice de l'utilisateur. Cependant, les développeurs de logiciels désirant profiter de cet environnement «Super VGA» sont confrontés à de sérieux problèmes.

En effet, par suite du manque de standards, ces produits sont conçus suivant des architectures très variables. Comme ils ne présentent pas d'interface logicielle standard, le développement des pilotes logiciels correspondants est techniquement très difficile et donc très coûteux, sauf quand il est réalisé directement par les fabricants des cartes qui se contentent, pour la plupart, de fournir les pilotes pour Windows, Autocad et Lotus 1-2-3. Rares sont donc les applications qui exploitent les possibilités de ces adaptateurs Super VGA.

Informations sur l'environnement vidéo

Pour l'instant, il n'existe pas de mécanisme standard permettant à une application de déterminer les caractéristiques matérielles de l'adaptateur Super VGA présent. Ce n'est qu'en prenant connaissance du détail des particularités de ce produit qu'il est possible d'informer l'application de l'identité de la carte avec laquelle elle travaille. Cette information ne peut s'obtenir le plus souvent qu'en allant tester et lire des registres situés à des adresses qui sont spécifiques à chaque fabricant OEM (Original Equipment Manufacturer).

Par conséquent, dans l'ignorance du type de carte installée, une application ne peut tirer parti des fonctions étendues de la carte. L'Extension du Bios Vesa VGA comporte plusieurs fonctions donnant des informations sur l'environnement vidéo, soit :

- **Fonction 00h** : information générale au niveau système, y compris une chaîne de caractères identifiant le constructeur, ainsi qu'un pointeur indiquant les modes vidéo supportés.
- **Fonction 01h** : utilisable par l'application pour obtenir des informations sur chacun des modes vidéo supportés.
- **Fonction 03h** : indication du mode vidéo courant.

Les différences entre les adaptateurs Super VGA se détectent souvent dans l'initialisation d'un mode vidéo et la pagination de la mémoire.

La configuration des registres d'entrée/sortie, l'emplacement du tampon vidéo dans l'espace d'adressage de l'unité centrale, l'emplacement et les fonctions du DAC, etc., demeurent en revanche conformes au standard VGA. L'Extension du Bios Vesa VGA comporte les fonctions d'interface correspondantes :

- **Fonction 02h** : fixe le mode de fonctionnement vidéo du Super VGA. L'application devient ainsi transparente à la procédure de fixation de ce mode vidéo.
- **Fonction 05h** : interface de pagination de la mémoire video.
- **Fonction 04h** : permet de sauvegarder et de rétablir un état Super VGA de manière transparente à l'application.

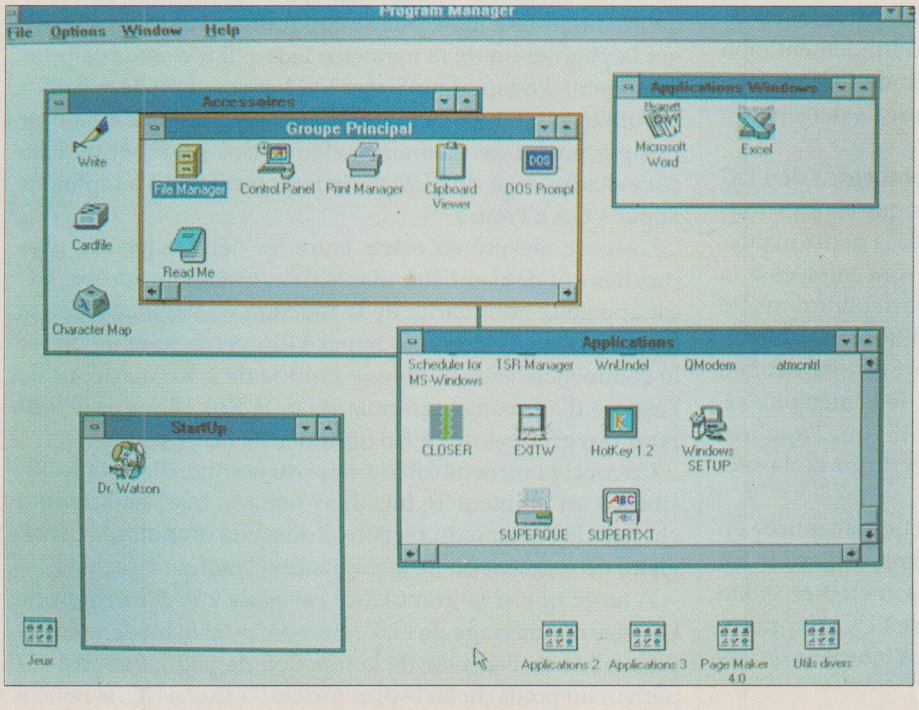
L'un des objectifs majeurs de l'Extension du Bios Vesa VGA vise à conserver la plus grande compatibilité possible avec l'environnement standard VGA. En aucun cas les extensions du Bios ne doivent compromettre cette compatibilité de fonctionnement. Les modifications à effectuer sur le Bios VGA exis-tant doivent être également minimales. Les extensions doivent pouvoir être implémentées sous forme de RAM ou de ROM.

La majorité des fabricants de cartes compatibles VGA ont tiré parti des possibilités des moniteurs multisynchrones en dotant leurs cartes de résolutions étendues. Plusieurs d'entre eux se sont regroupés pour créer le comité Vesa (Video Electronics Standard Association), dans le but de normaliser la gestion logicielle de ces nouvelles r ésolutions par le biais du Bios repris par la plupart des cartes super VGA d'aujourd'hui.

Les logiciels graphiques gérant un contrôleur Super VGA le feront de la même manière que pour un VGA standard, c'est-à-dire en écrivant directement dans une mémoire tampon, en manipulant des registres de contrôle graphique, en programmant directement la palette de couleurs, etc. Aucune fonction graphique importante ne devra être exécutée par le matériel.

C'est pour cette raison que l'Extension du Bios Vesa VGA ne comporte aucune fonction graphique du type BitBit, dessin d'une ligne ou d'un cercle, etc. Est également exclu du domaine de l'extension le traitement des différents types de moniteurs et leur timing. Ces sujets sont traités dans d'autres spécifications Vesa. Il est important de noter que les fonctionnalités qui peuvent être placées dans l'Extension du Bios Vesa VGA sont fortement restreintes par la place disponible dans les ROM de certaines implémentations existantes.

Le mode 1 024 x 768 est de loin le plus intéressant, surtout sous Windows, mais il demande l'achat d'un moniteur adéquat.



Le Super VGA et le Bios VGA standard

L'extension Vesa touche à deux fonctions standard du Bios VGA : 00h (fixe le mode VGA à utiliser) et 0Fh (lecture de l'état vidéo courant). Elles sont remplacées respectivement par les fonctions Vesa 02h (fixe le mode Super VGA à utiliser) et 03h (lecture du mode Super VGA courant).

Les applications antérieures à Vesa (programmes «POP UP et TSR, Terminate Stay Resident» anciens, ou la commande CLS du MS-DOS) peuvent continuer à utiliser 0Fh pour lire le mode vidéo courant et ensuite 00h pour rétablir et réinitialiser le mode vidéo antérieur. Il est recommandé pour cela que le numéro de mode vidéo donné par la fonction VGA 0Fh puisse être réutilisé par la fonction VGA 00h afin de rétablir le mode vidéo antérieur. Ce qui signifie que le Bios doit mémoriser l'identité du dernier mode Super VGA utilisé. Il est recommandé, mais non obligatoire, que les modes Super VGA supportent les fonctions de sortie (telles que sortie TTY, défilement, pixel, etc.). Dans le cas contraire, le bit D2 du champ ATTRIBUTS DE MODE ne doit pas être utilisé.

Les numéros d'identification des modes Super VGA

Les numéros de mode vidéo en VGA comportent 7 bits dont les 20 valeurs numérotées de 00h à 13h sont utilisées. Les extensions de 14h à 7Fh employées par les constructeurs utilisent les 6 bits de poids le plus faible. Le septième bit est réservé à la fonction Bios 00h pour indiquer s'il faut ou non effacer la mémoire vidéo. Pour s'affranchir des limitations de la numérotation VGA, les numéros de mode Vesa comportent 15 bits

(32 768 indications possibles). Ils sont passés à la fonction Vesa 02h par le registre BX. Les 8 bits de poids le plus faible (BL) donnent le numéro du mode (256 modes possibles); l'état du huitième bit spécifie l'origine du mode graphique (s'il est à 0, il s'agit d'un mode VGA, sinon d'un mode Vesa). Les bits 9 à 15 (BH) sont réservés à un usage ultérieur. Les numéros VGA et leur extension OEM sont donc des sous-ensembles du Vesa, caractérisés par BH à 0.
Les numéros Vesa actuellement définis sont :

N° 15 bits	N° 7 bits	Résolution	Couleurs
100h	-	640 x 400	256
101h	-	640 x 480	256
102h	6Ah	800 x 600	16
103h	-	800 x 600	256
104h	-	1024 x 768	16
105h	-	1024 x 768	256
106h	-	1280 x 1024	16
107h	-	1280 x 1024	256

Les fenêtres de mémoire vidéo

Un sous-système standard VGA utilise 256 kilo-octets de mémoire vidéo et comprend un mécanisme d'adressage de cette mémoire. Les modes Super VGA simples ou étendus ont besoin de plus de 256 Ko de mémoire, mais pour des raisons de compatibilité, ils doivent faire appel au système d'adressage n'utilisant que l'espace standard prévu à cet effet.

Les fenêtres de mémoire vidéo de l'unité centrale (UC) donnent les moyens de le faire. Ce chapitre décrit le fonctionnement de différents types de matériels d'implémentation du système de fenêtres de mémoire vidéo de l'UC et leur relation aux modèles logiciels proposés par l'extension du Bios Vesa VGA.

Les fonctions Vesa de fenêtres de mémoire vidéo de l'UC ont été conçues pour ne mettre dans le Bios que les fonctions matérielles non standard, indépendantes de la performance. Les autres, asservies à la performance, sont laissées à la charge de l'application. Ceci assure leur portabilité dans les différents systèmes VGA ainsi que la rapidité de fonctionnement découlant d'un accès matériel direct. En particulier, c'est le Bios Vesa qui assure la pagination de la mémoire vidéo dans l'espace d'adressage de l'UC, tandis que l'application effectue les opérations de lecture/écriture de la mémoire.

Cette interface mixte matérielle/logicielle est assurée en communiquant à l'application les paramètres de gestion du mécanisme matériel de pagination de la mémoire vidéo dans l'espace d'adressage correspondant de l'UC. L'application gère alors la pagination en restant à l'intérieur de ces paramètres.

Le premier objectif de la mise en place de la mémoire vidéo étendue est de la rendre accessible aux logiciels d'application : l'espace d'adressage standard de l'UC pour le VGA en mode graphique 16 couleurs est placé au segment A000h et comprend 64 kilo-octets. Il donne accès aux 256 kilo-octets du standard VGA, c'est-à-dire 64 Ko par plan. L'accès à la mémoire vidéo étendue est obtenu en paginant des fractions de la mémoire vidéo dans l'espace d'adressage vidéo standard de l'UC.

Tous les équipements Super VGA doivent fournir aux logiciels un mécanisme permettant de spécifier le déplacement depuis le commencement de la mémoire vidéo à paginer au début de l'espace d'adressage de l'UC.

Concept de granularité de la fonction de pagination

La plupart des algorithmes logiciels sont organisés autour de deux boucles imbriquées : l'externe, pour le passage d'une ligne de balayage à l'autre, l'interne, pour le balayage de la ligne. C'est cette dernière boucle qui est le point critique de la performance des logiciels.

Suivant les dimensions du rectangle ou sa position à l'écran, la mémoire vidéo correspondante peut n'être paginée que partiellement dans l'espace d'adressage de l'UC, et l'accès au reste du rectangle demande un changement de pagination.

Il est souhaitable que le test de repagination se fasse en dehors de la boucle interne. Il faut donc sélectionner l'offset de pagination du commencement de la mémoire vidéo au début de l'espace d'adressage de l'UC, de telle manière qu'au moins une ligne puisse être traitée sans avoir à changer la pagination de la mémoire vidéo. Il n'existe pas pour le moment d'adaptateur Super VGA permettant de spécifier cet offset «on a byte boundary», et la manière de disposer l'emplacement de mémoire vidéo désiré au début de l'espace d'adressage de l'UC varie largement d'un adaptateur Super VGA à l'autre.

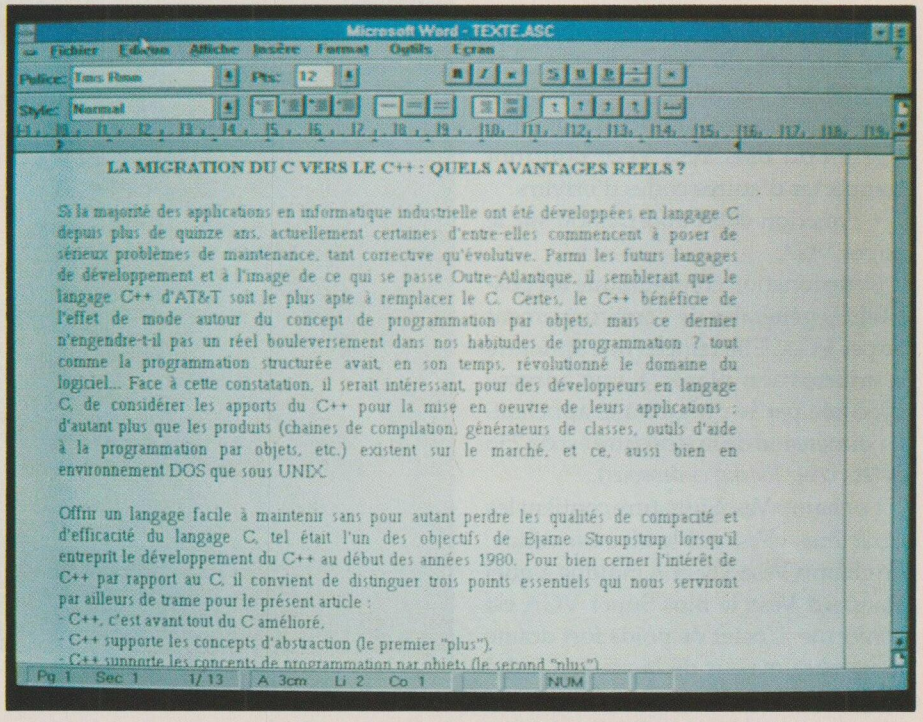
L'espace mesuré en octets entre les deux octets les plus proches qui peuvent être placés dans une seule adresse UC est appelé la granularité de la fonction de pagination de la fenêtre. Certains systèmes Super VGA permettent de placer le commencement de chaque groupe de 4 Ko au début de l'espace d'adressage (granularité de 4 Ko), d'autres le font pour des groupes de 64 Ko (granularité de 64 Ko).

Ce concept est semblable à celui du nombre d'octets accessibles à un pointeur 16 bits dans une UC Intel sans avoir à changer le contenu du registre d'offset (la granularité du registre de segment ou de la pagination étant de 16 octets).

A noter que si la granularité est égale à la dimension de l'espace d'adressage de l'UC, c'est-à-dire si le bit de poids le plus faible de l'adresse de la fonction de pagination est supérieur au poids du bit le plus fort de l'adresse UC, la boucle

interne de balayage d'une ligne devra contenir le test de traversée de début ou de fin de l'espace d'adressage de l'UC.

Autrement dit, si la longueur de l'espace d'adressage (la granularité dans ce cas) n'est pas un multiple de la longueur de balayage, la dernière ligne de balayage se trouvera dans deux espaces mémoire vidéo ne pouvant être paginés simultanément dans l'espace d'adressage de l'UC.



Le mode 800 x 600 devient pratiquement indispensable pour un travail confortable sous Windows.

Utilisation de données provenant de zones différentes

Il est quelquefois nécessaire ou utile de déplacer des données provenant de deux zones différentes de la mémoire vidéo. C'est, par exemple, le cas de la mémorisation de menus dans une zone de la mémoire vidéo au-delà de celle affichée. Il faut, dans ce cas, utiliser deux fenêtres paginées séparément si la distance entre la zone de départ et celle d'arrivée est supérieure à la taille de la fenêtre vidéo de l'UC. Cela revient à déplacer les données d'une fenêtre à l'autre, donc à lire dans la première et à écrire dans la seconde.

Il peut être utile de pouvoir lire dans les deux fenêtres et écrire dans la fenêtre de destination. Ce cas se présente lors de la création d'une image dans la zone de destination par addition de l'image de la zone départ à celle située originalement dans la zone arrivée.

L'extension du Bios Vesa VGA n'impose pas une solution unique à l'implémentation matérielle des fenêtres de mémoire vidéo UC.

Les informations nécessaires à la compréhension de la solution matérielle utilisée sont passées aux applications par l'intermédiaire des fonctions du Bios. Il y a trois types de base du fenêtrage matériel, présentés ci-dessous. Ces types sont indépendants de la granularité. A noter également qu'un système VGA peut utiliser 128 Ko d'espace d'adressage CPU, segment A000h.

• Système à une seule fenêtre

Certains systèmes n'utilisent qu'une seule fenêtre. Celle-ci doit être accessible en écriture et en lecture. Cette solution entraîne une dégradation importante des performances lors du déplacement de données en mémoire vidéo d'une distance uniquement à la dimension de l'espace d'adressage de l'UC.

• Système à deux fenêtres

De nombreux systèmes Super VGA utilisent deux fenêtres pour rendre plus aisé le déplacement des données dans la mémoire vidéo. Il existe deux manières de le faire.

a) les fenêtres superposées

Certains systèmes matériels distinguent deux fenêtres par le type d'opération -écriture ou lecture- que veut effectuer l'UC. Dans ce cas, les fenêtres peuvent partager -et la plupart du temps, elles le font- le même espace d'adressage UC. Une fenêtre sert uniquement à l'écriture et l'autre à la lecture.

b) les fenêtres non superposées

Elles se distinguent par leur adresse dans l'espace d'adressage de l'UC. Dans ce cas, les deux peuvent servir en écriture ou en lecture.

Le Bios VGA étendu

Les modes Super VGA sont supportés par plusieurs nouveaux appels du Bios. Pour un maximum de compatibilité avec le Bios VGA standard, ces appels sont regroupés sous le numéro de fonction 47H. Ce numéro est passé à l'aide de l'interruption 10H par l'intermédiaire du registre AH. Ce numéro de fonction n'est utilisé par aucun des Bios VGA. Chaque fonction retourne une information d'état dans le registre AX en utilisant le format ci-dessous :

```
AL ==  4Fh : La fonction est supportée
AL ==  4Fh : La fonction n'est pas supportée
AH ==  00h : Appel de la fonction réussi
AH ==  01h : Appel de la fonction infructueux
```

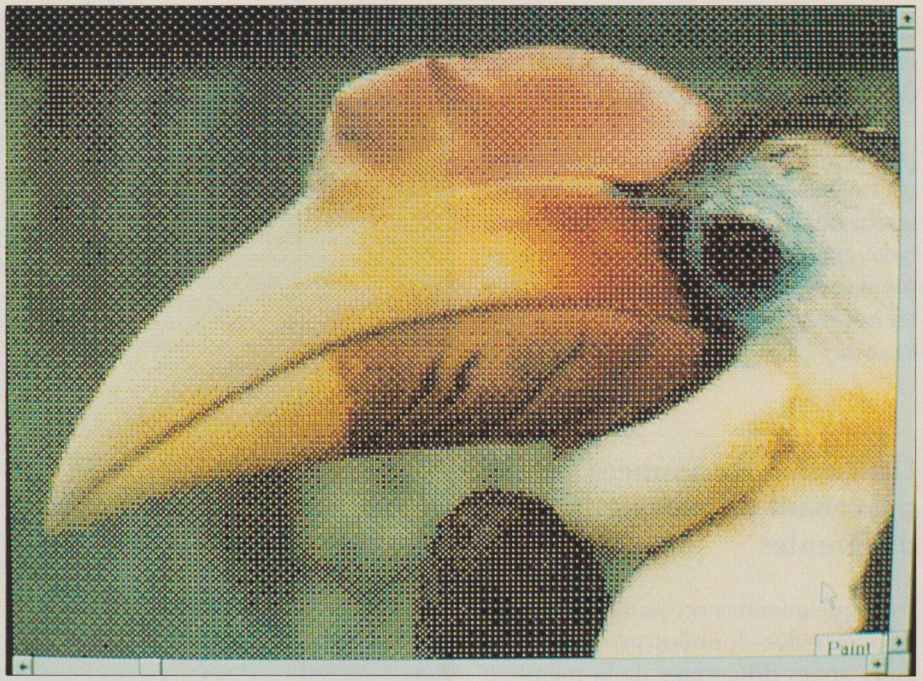
VESA : MODE D'EMPLOI

Les logiciels doivent traiter une valeur différente de zéro du registre AH comme une condition d'erreur générale. Les versions ultérieures de l'extension du Bios Vesa VGA pourront comporter d'autres codes d'erreurs.

• **Fonction 00h** : information sur le Super VGA

Cette fonction communique les possibilités générales de l'environnement Super VGA. Elle remplit une structure d'information placée à une adresse spécifiée par le programme appelant. La dimension de la structure est de 256 octets (*voir listing ci-dessous*).

Le champ VesaSignature contient les caractères «Vesa» si le bloc est valide. Le champ Vesa Version indique à quel standard Vesa le Bios Super VGA est conforme. L'octet de poids fort donne le nombre majeur de la version, celui de poids faible le nombre mineur. Le numéro initial est 1.0. Les applications écrites à partir des particularités d'une version donnée de l'extension du Bios Vesa sont garanties fonc- tionner avec les versions ultérieures. Les extensions Bios Vesa auront une compatibilité ascendante complète.



Comme le montrent ces deux photos de Toucan, le passage du mode 640 x 480 seize couleurs en 256 couleurs améliore considérablement la qualité de l'image, mais au détriment de la vitesse d'affichage et de la taille des fichiers graphiques qui quadruple.

```
Entrée : AH = 4Fh   Le Super VGA est
                    supporté
         AL = 00h   Les informations sur
                    le Super VGA ont été
                    communiquées
Sortie : AX = Statut
```

Tous les autres registres sont inchangés.

Structure de l'information :

```
VgainfoBlock    struc

VESASignature   db 'VESA'     ; signature
                              sur 4 octets
VESAVersion     dw ?          ; numéro de la
                              version VESA
OEMStringPtr    dd ?          ; pointeur
                              vers la
                              chaîne de
                              caractères
                              OEM
Capabilities    db 4 dup(?)   ; possibilités
                              de l'
                              environnement
                              vidéo
VideoModePtr    dd ?          ; pointeur des
                              modes VGA
                              supportés.

VgaInfoBlock    ends
```

Le champ OEMStringPtr est un pointeur long vers une chaîne de caractères définie par le fabricant OEM. Elle est terminée par 0. La chaîne de caractères, de format libre, peut être utilisée pour identifier le circuit intégré vidéo, la carte vidéo et la configuration de la mémoire d'un driver moniteur matériel spécifique.

Le champ Capabilities décrit les particularités générales supportées par l'environnement vidéo. Le pointeur VideoModePtr pointe vers une liste de numéros de mode supportés par le Super VGA (Vesa ou OEM). Chaque numéro comprend un mot de 2 octets.

La liste est terminée par -1 (0FFFFh). La liste de ces numéros est donnée plus loin. L'adresse pointée peut être en ROM si elle est fixe, en RAM si elle est générée dynamiquement lors de l'exécution du programme.

• **Fonction 01H** : informations sur le mode Super VGA.

Cette fonction retourne des informations sur un mode spécifique Super VGA. Celles-ci sont données dans un bloc de 256 octets, à l'adresse spécifiée par l'appelant et suivant une structure déterminée.

Le numéro du mode Vesa utilisé donne déjà implicitement toute une série d'informations. Mais certaines implémentations Super VGA peuvent utiliser des modes autres que ceux prévus par Vesa. Cette fonction retourne d'autres informations permettant d'utiliser ces modes.

ModeAttributes donne quelques caractéristiques impor-

```
Entrée :   AH = 4Fh Support Super VGA
           AL = 01h Appelle la fonction 01h
           CX = Numéro du mode Super VGA
                concerné
           ES:DI  Adresse de l'information
                  demandée

Sortie :   AX = Statut
```

Tous les autres registres restent inchangés

Structure du bloc d'information :

```
ModeInfoBlock     struc   ;  Information
                            obligatoire

ModeAttributes    dw ?    ; Mode attributs
WinAAttributes    db ?    ; Attributs de
                            la fenêtre A
WinBAttributes    db ?    ; Attributs de la
                            fenêtre B
WinGranularity    dw ?    ; Granularité de
                            la fenêtre
WinSize           dw ?    ; Dimensions de
                            la fenêtre
WinASegment       dw ?    ; Début segment
                            de la fenêtre A
WinBSegment       dw ?    ; Début segment
                            de la fenêtre B
WinFincPtr        dd ?    ; Pointeur
                            fonction
                            fenêtre
BytesPerScanLine  dw ?    ; Info étendue
                            sur les octets
                            par ligne
                  ; Information
                    facultative
XResolution       dw ?    ; Résolution
                            horizontale
YResolution       dw ?    ; Résolution
                            verticale
XCharSize         db ?    ; Largeur
                            cellule
                            caractère
YcharSize         db ?    ; Hauteur
                            cellule
                            caractère
NumberOfPlanes    db ?    ; Nombre de
                            plans mémoire
BitsperPixel      db ?    ; Nombre de bits
                            par pixel
NumberOfBanks     db ?    ; Nombre de
                            bancs
MemoryModel       db ?    ; Type de modèle
                            mémoire
BankSize          db ?    ; Dimension du
                            banc en Kilo-
                            octets

ModeInfoBlock     ends
```

tantes du mode Vidéo (si les bits sont à 1, l'information correspondante est supportée) :

• D0 indique que le mode demandé peut être utilisé avec la configuration vidéo existante.

Cette information est utile lorsque ce mode demande un certain type de moniteur qui n'est pas connecté.

• D1 indique si l'information sur le mode étendu est disponible. Les modes vidéo Vesa ont certaines caractéristiques connues, telles que la résolution, le nombre de plans, le format du pixel, etc. Par suite de l'importante limitation de place dans les implémentations ROM de l'extension du Bios Vesa VGA, il doit être possible de donner par ailleurs cette information sur un mode Vesa.

• D2 indique si le Bios supporte les fonctions de sortie telles que TTY, défilement, pixel, etc., dans le mode demandé. (Il est recommandé mais pas obligatoire que le Bios puisse supporter toutes les fonctions de sortie).

• D3 indique si le mode est monochrome (0) ou couleur (1). Les modes monochromes ont leur adresse CRTC en 3D4h. Seuls les bits 3 (vidéo) et 4 (intensité) de l'attribut du gestionnaire sortie sont significatifs. Le mode texte monochrome a les attributs Off, vidéo, surintensité, clignotant, etc. Le mode graphique monochrome est à deux plans et dispose des mêmes attributs. Les modes étendus bicolores ont leur adresse CRTC en 3D4h. Ils ne comportent qu'un seul plan et un bit par pixel. Dans cette classification, les modes standard VGA 06h et 11h sont des modes couleur, 07h et 0fh des modes monochromes.

• D4 type de mode. 0 mode texte, 1 mode graphique.

• D5 à D15 : réservé.

BytesperScanline indique le nombre d'octets par ligne logique. La ligne logique peut être de taille égale ou supérieure à la ligne de balayage.

WinAAttributes et WinBAttributes indiquent si la fenêtre existe et peut être lue et écrite.

Signification des bits :
- D0 Si égal à 0 la fenêtre n'est pas supportée.
- D1 Si égal à 0 la fenêtre peut être lue. - D2 Si égal 0 la fenêtre peut être écrite. - D3 à D7 : réservé.

Wingranularity donne la granularité de la fenêtre en kilo-octets. WinSize donne la taille en kilo-octets.

Winxsegment indique l'adresse des segments où les fenêtres sont placées dans l'espace d'adressage UC.

WinFuncAddr indique l'adresse de la fonction fenêtre vidéo de l'UC. Cette fonction peut être appelée directement ou par l'intermédiaire de la fonction Bios Vesa 05h. L'appel direct donne un accès plus rapide aux registres de pagination matériels.

X,YResolution indique la largeur et la hauteur du mode vidéo. En mode graphique, l'unité est le pixel, en mode caractère, l'unité est le caractère. La résolution en mode caractère peut être exprimée en pixels en multipliant celle donnée en caractères par le nombre de pixels d'une cellule.

X,YCharCellSize donne la taille en pixels de la cellule caractère.

NumberOfPlanes indique le nombre de plans mémoire disponibles.

BitsPerPixels spécifie le nombre de bits définissant la couleur d'un pixel. Sa valeur est 4 pour 16 couleurs et 8 pour 256. Les organisations non stan-dard sont définies par le nombre de plans et le nombre de pixels. Par exemple, un mode

compacté 16 couleurs est décrit par 1 plan et 4 bits par pixel.

MemoryModel indique le type général d'organisation mémoire utilisé dans le mode à l'aide de la codification suivante :

```
00h Mode test
01h graphique CGA
02h graphique Hercules
03h planaire 4 plans
04h pixel compacté
05h Non-chaîné 4, 256 couleurs
06h-0fh Réservé à VESA
10h-ffh Réservé aux OEM
```

NumberofBanks retourne le nombre de bancs (groupes de lignes). Le reste de la division du numéro de la ligne par le nombre de bancs donne le numéro du banc où se trouve cette ligne, et le quotient son numéro dans ce banc. Les modes graphiques CGA ont deux bancs, les modes graphiques Hercules quatre.

Pour les modes qui n'utilisent pas la notion de bancs (modes VGA 0Dh-13h), ce champ doit être mis à 1.

BankSize donne en kilo-octets la taille d'un banc. En mode CGA et Hercules, il est égal à 8, chaque banc mesurant 8192 octets. Pour les modes qui n'utilisent pas la notion de bancs (modes VGA 0Dh-13h), ce champ doit être mis à 0.

• **Fonction 02H** : fixation du mode vidéo Super VGA à utiliser

Cette fonction installe un mode vidéo Super VGA. Si le mode ne peut être installé, le Bios doit laisser inchangé l'environnement vidéo et retourner un code d'échec.

```
Entrée  : AH = 4Fh Mode Super VGA
          AL = 02h Installe un mode
                   vidéo Super VGA
          BX = Numéro du mode vidéo
               demandé
             D0-D14 = mode vidéo demandé
             D15    = Flag d'effacement
                      de la mémoire
                    0 = effacer la
                        mémoire
                    1 = ne pas effacer
                        la mémoire

Sortie  : AX = Statut (succès ou échec)
```

Tous les autres registres sont inchangés.

- fonction 03h : mode vidéo courant

Cette fonction retourne le numéro du mode vidéo courant dans le registre BX.

```
Entrée :  AH = 4Fh Mode Super VGA
          AL = 03h Demande du numéro du
                   mode vidéo courant

Sortie :  AX = Statut (échec ou succès)
          BX = Numéro du mode vidéo.
```

Le Bios VGA standard donne cette information par la fonction 0Fh dans le registre AL, qui contient également un bit de flag indiquant si le mode avait été établi en effaçant ou non la mémoire.

La fonction 03h du Bios Super VGA ne retourne pas ce flag.

• **Fonction 04H** : Sauvegarde/rétablissement de l'état vidéo Super VGA

Cette fonction se subdivise en trois sous-fonctions permettant de sauvegarder ou restaurer l'état vidéo. Elle est l'équivalent de la fonction Bios VGA 1Ch qu'elle remplace pour des raisons de compatibilité, cette fonction ne contenant pas

```
Entrée : AH = 4Fh Mode Super VGA
         AL = 04h Sauvegarde/restauration
                  de l'état vidéo
         DL = 00h Retourne la taille du
                  buffer de
                  sauvegarde/rétablis.
         CX = Etats demandés
              D0 = Vidéo matériel
              D1 = Données BIOS vidéo
              D2 = Etat DAC vidéo
              D3 = Etat Super VGA

Sortie : AX = statut de la requête
         BX = Nombre de blocs 64 octets
              mémorisant les
              indications d'état
```

Tous les autres registres sont inchangés.

```
Entrée :  AH = 4Fh Mode Super VGA
          AL = 04h Sauvegarde /
                   restauration de l'état
                   vidéo
          DL = 01h Sauvegarde de l'état
                   vidéo
          CX = Etats demandés (voir plus
               haut)
          ES:BX= Pointeur vers le buffer

Sortie :  AX = Statut de la requête
```

Tous les autres registres sont inchangés.

```
Entrée :  AH = 4Fh Mode Super VGA
          AL = 04h Sauvegarde /
                   restauration de l'état
                   vidéo
          DL = 02h Restaure l'état vidéo
          CX = Etats demandés (voir plus
               haut)
          ES:BX= Pointeur vers le buffer

Sortie :  AX= Statut de la requête
```

l'espace nécessaire à la définition des états Super VGA. Cette fonction fixe ou lit dans la mémoire vidéo la position de la fenêtre demandée. Elle donne un accès direct aux registres de pagination matériels.

Cette fonction s'utilise en tandem avec la fonction Super VGA 01h qui donne la dimension, l'emplacement et la granularité des fenêtres.

```
Entrée :   AH = 4Fh Mode Super VGA
           AL = 04h Gestion de la fenêtre
                    mémoire vidéo SuperVGA

           BH = 00h Choix de la fenêtre
           BL = Type de fenêtre
                0= Fenêtre A
                1= Fenêtre B
           DX = Emplacement de la fenêtre
                dans la mémoire vidéo,
                exprimée en unité de
                granularité

Sortie :   AX = Statut

Entrée :   AH = 4Fh Mode Super VGA
           AL = 04h Gestion de la fenêtre
                    mémoire vidéo
           BH = 01h Demande la position de
                    la fenêtre
           BL = Type de fenêtre
                0= Fenêtre A
                1= Fenêtre B

Sortie :  AX = Statut
          DX = Emplacement de la fenêtre
               dans la mémoire vidéo,
               exprimée en unité de
               granularité
```

Cette fonction peut être utilisée dans un appel de type FAR (comportant un changement de segment).

L'adresse de la Fonction Bios s'obtient par la fonction Vesa 01h. Un champ du ModeInfoBlock contient l'adresse de cette fonction. Cette fonction peut ne pas être la même dans différents modes vidéo d'une implémentation spécifique du Bios. Le pointeur de la fonction doit donc pouvoir être connu pour chaque mode. Dans le cas d'un appel de type FAR, aucune information de statut n'est donnée. De même, le contenu des registres AX et DX sera modifié. S'il faut préserver ce dernier, l'application doit prendre les mesures nécessaires avant l'appel de la fonction afin de sauvegarder leur contenu.

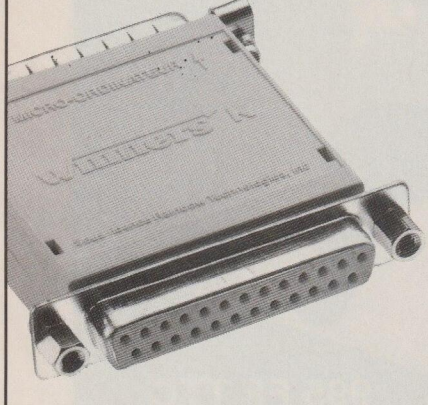
Exemple d'application

Cet exemple illustre comment une application s'interface avec l'extension Bios Vesa. L'application est conçue non seulement pour les modes vidéo Vesa dont elle appelle les fonctions, mais également pour d'autres modes. Elle doit donc s'informer des modes vidéo disponibles avant de fixer ceux qu'elle utilisera.

WINNERS K

LA PROTECTION ÉLECTRONIQUE

Winners, leader français de la duplication, présente : les clefs de protection électroniques "WINNERS K".



La gamme des clefs "WINNERS K" constitue aujourd'hui le système de protection absolu du logiciel. Plus de 50 interfaces sont disponibles pour les langages évolués les plus courant. Les clefs "WINNERS K" sont compatibles avec les systèmes DOS/OS/2, WINDOWS, UNIX, XENIX, en version monoposte ou réseau ; un modèle spécial étant destiné à la protection des logiciels MACINTOSH.

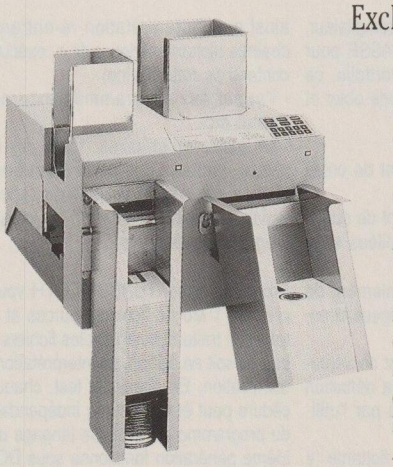
winners
(1) 40 24 05 85
114, rue des Pyrénées - 75020 PARIS
Fax : (1) 40 24 28 25 - Télex : 214306 F

ND CONSEIL

VENTUNO

LE MAITRE DUPLICATEUR

Exclusif !!... Le Switch 2 Combo :



Unique machine au monde permettant la duplication simultanée des deux standards 5"1/4 et 3"1/2 ; le Switch 2 Combo existe en version "CONNECTABLE" (applications techniques) et en version "AUTONOME" (applications bureautiques).

Ventuno Inc. propose une gamme complète de robots de duplication autonomes ou connectables pour IBM PC/PS, compatibles et MACINTOSH, permettant le traitement de tous formats physiques et logiques dans les environnements Dos, OS/2, Windows, UNIX, AMIGA, ATARI...

Les matériels Ventuno sont garanties 1 an pièce et main d'œuvre.
Ils sont distribués en exclusivité par :

winners
(1) 40 24 05 85
114, rue des Pyrénées - 75020 PARIS
Fax : (1) 40 24 28 25 - Télex : 214306 F

ND CONSEIL

Service lecteurs n° 8 (dernière page)

VESA : MODE D'EMPLOI

1) L'application détermine d'abord un buffer de 256 octets.

Celui-ci sera utilisé par le Bios Vesa pour retourner les informations nécessaires sur l'environnement vidéo. Cette allocation peut être fixe ou dynamique et temporaire.

2) L'application vérifie ensuite la présence du Bios Vesa étendu en appelant la fonction Vesa Bios 00h.

Si celle-ci ne met pas le registre AX à 004Fh (Bios Vesa absent), l'application doit prendre les mesures nécessaires. Dans le cas contraire, la fonction a placé les informations convenues dans le buffer, ce qui est signalé par les caractères «Vesa» au début du bloc.

Le champ VesaVersion indique les fonctionnalités disponibles du Bios Vesa étendu, le champ OEMStringPtr les informations spécifiques au fabricant OEM de la carte vidéo, le champ VideoModePtr renvoie à la liste des modes vidéo supportés par l'environnement.

3) L'application crée alors un nouveau buffer pour recevoir, à l'aide de la fonction Vesa 01h, les informations sur les modes vidéo supportés.

En utilisant les informations données par le VideoModePtr dans la phase 2, l'application appelle la fonction 01h avec le numéro de mode adéquat jusqu'à ce qu'un mode approprié soit trouvé.

Dans le cas contraire, l'application doit prendre les mesures nécessaires.

La fonction 01h place les informations requises sur le mode demandé dans le nouveau buffer.

Le champ ModeAttributes indique au bit D0 si le mode demandé est supporté ou non (dans ce cas D0 est à 0) par le matériel, par exemple si le moniteur spécifique à ce mode est bien connecté.

4) Le pas suivant consiste à prendre les mesures nécessaires pour sauvegarder l'identité du mode en cours, afin de pouvoir le rétablir en fin d'application.

Celle-ci, si le mode en cours est Super VGA (VGA standard ou OEM), est donnée par la fonction Vesa 03h. Sinon cette dernière met à 0 l'octet de poids le plus élevé.

5) Le mode vidéo désiré est initialisé à l'aide de la fonction Vesa 02h.

L'application reçoit alors un accès complet au matériel et à la mémoire VGA.

6) En fin d'application, celle-ci rétablit le mode antérieur à l'aide de la fonction 02h. □

DANS LA JUNGLE DES LANGAGES DE DEVELOPPEMENT, VOICI DES RAISONS DE FAIRE LE BON CHOIX

TURBO-FORTH

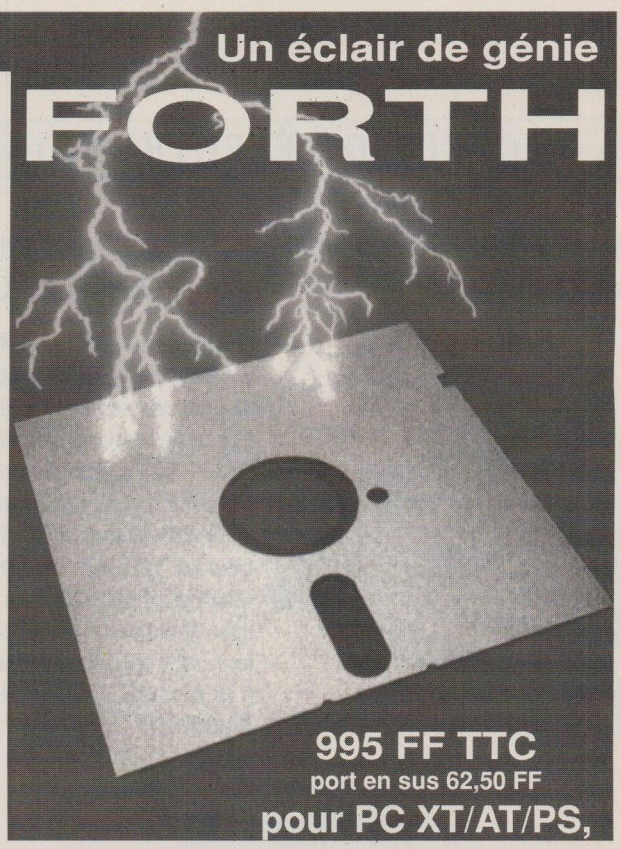
Un éclair de génie

VOUS OFFRE:

- à la fois un interpréteur et un compilateur, fonctionnant EN UNE SEULE PASSE pour IBM PC/XT/AT/PS2, et Atari Portfolio, ce sans nécessiter l'utilisation de code objet et de linker!!!
- un vrai multi-tâche sous DOS,
- un métacompilateur permettant de créer des exécutables hyper-compacts,
- un débogueur-traceur permettant de suivre pas à pas les procédures compilées et le contenu de la pile de données,
- un décompilateur capable également de désassembler les procédures rédigées directement en code machine.
- un assembleur structuré utilisant de véritables mnémoniques, permettant la définition de nouvelles macro-instructions par l'utilisateur,
- trois modules de calcul virgule flottante (y compris un modèle CORDIC)
- une auto-documentation intégrée et contextuelle,
- un éditeur offrant des fonctions traitement de texte,
- le traitement hiérarchisé des fichiers sources ASCII permettant huit niveaux d'imbrication, ainsi que l'interprétation ré-entrante des chaînes alphanumériques (e.g. exécution du contenu de cette chaîne)
- l'appel aux programmes externes: C, Pascal, etc...
- de nombreux utilitaires: accès aux fichiers dBase, pilotage des ports d'entrée/sortie, gestion des modes graphiques, accès aux variables d'environnement DOS, création de programmes résidents.

Produit français, TURBO-FORTH vous offre près de 1 Mo de fichiers sources et exécutables. Il traite directement les fichiers ASCII, que ce soit en édition, en interprétation, et en compilation. En phase de test, chaque procédure peut être exécutée indépendamment du programme général. Ce langage de quatrième génération fonctionne sous DOS (2.0 et suivants) et sous CONCURRENT DOS 386. Il est accompagné de deux manuels (manuel d'apprentissage et guide de référence).

Configuration minimale requise : 128 Ko RAM (64 Ko pour la version ATARI Portfolio)!!!!



995 FF TTC
port en sus 62,50 FF
pour PC XT/AT/PS,

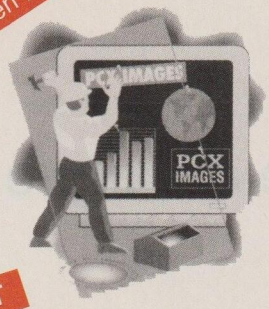
REM CORP - 17, rue de la Lancette - 75012 PARIS (France)
tel: (33) 1- 43.42.32.15 fax: (33) 1- 43.40.96.53 Minicom: 3612 REM CORP 143409653 Calvacom: cité FORTH

Service lecteurs nº 9 (dernière page)

RAJOUTEZ LA PUISSANCE DU GRAPHISME A VOS APPLICATIONS

2 490 FHT
2 953,14 F TTC
manuel en français

PCX PROGRAMMER'S TOOLKIT 4.0



- Plus de 90 fonctions graphiques
- Possibilité d'affichage de graphismes dans toutes les applications au format PCX
- Fonctions de sauvegarde, d'impression, de manipulation d'images
- Fonctions de capture d'écrans, de gestion de fichiers image
- Possibilité de mélanger textes et graphismes
- Compatible avec 12 compilateurs et 6 langages ; support EMS ; 22 modes vidéo supportés
- Pas de royalties ; version avec source disponnible, version française

1 290 F HT

GX TEXT 2.0

(1529,94 F TTC)

Permet d'écrire des textes tout en étant en mode graphique. Le mélange des images et des textes créés devient très facile. Un éditeur de fontes est inclus pour compléter plus de 70 fontes livrées en standard. Version US.

1 690 F HT

GX EFFECTS

(2003,34 F TTC)

Ajoutez des effets spectaculaires à vos programmes : effet de diaporama, explosion de l'image, affichage de l'image en diagonale... Un sous programme musical permet de mélanger musique et image ! Version US.

1 690 F HT

GX GRAPHICS

(2003,34 TTC)

Une librairie graphique portable qui offre toutes les fonctions graphiques incluses dans le BGI (Borland) ou les MS Librairies mais qui est compatible avec 12 compilateurs.

Service lecteurs n° 10 (dernière page)

INNOSOFT **Tél. (1) 40 99 28 00 - Fax : 40 99 28 88**

BON DE COMMANDE ET DE DOCUMENTATION à renvoyer à INNOSOFT - 2, rue des Bourets - 92150 Suresnes

I_I Je commande I_I Je désire recevoir des documentation sur vos produits : .....

I_I PCX PROGRAMMER'S TOOLKIT. 4.0 au prix unitaire de **3024,30 F TTC**
(soit 2953,14 F TTC + 71,16 F TTC de port)
I_I GX TEXT 2.0 au prix unitaire de **1601,10 F TTC**
(soit 1529,94 F TTC + 71,16 F TTC de port)
I_I GX EFFECTS au prix unitaire de **2074,50 F TTC**
(soit 2003,34 F TTC + 71,16 F TTC de port)
I_I GX GRAPHICS au prix unitaire de **2074,50 F TTC**
(soit 2003,34 F TTC + 71,16 F TTC de port)

Ci-joint mon règlement :
I_I Chèque I_I Carte Bleue N°... ..... Date d'expiration.....
I_I Contre-remboursement (60 F TTC de frais en sus)

Nom : .....
Société : .....
Adresse :.....
Code postal : I_I_I_I_I_I Ville : .....

APPRENDRE WINDOWS

Intégrer les contrôles et les boîtes de dialogue

MARC ADLER

Les boîtes de dialogue et les contrôles qu'elles contiennent font partie intégrante de l'environnement graphique de Microsoft Windows. Ce sujet apparaît assez vaste pour qu'on lui consacre deux articles : le premier figurant dans le numéro 4 de Langages & Systèmes, le second se trouvant à présent exposé. D'abord, nous achèverons notre panorama des contrôles en examinant les boîtes combos (ou combinées), les barres de défilement et les contrôles définis par l'utilisateur. Puis nous placerons ceux-ci dans leur contexte, c'est-à-dire dans les boîtes de dialogue. Nous terminerons en ajoutant quelques-unes de ces dernières à notre application de démonstration (un portefeuille d'actions).

Les boîtes combos

La boîte combo, introduite dans la version 3.0 de Windows, «combine» en une seule fenêtre un contrôle d'édition (champ d'édition) monoligne et une boîte liste. Elle peut être utilisée pour entrer une valeur dans un champ d'édition ou pour choisir une chaîne dans une liste prédéfinie. Ainsi, une boîte combo est susceptible de remplacer un groupe de boutons radio dans une boîte de dialogue, car elle prend moins de place. Dans un traitement de texte, vous pouvez spécifier, en donnant le nombre de points, la taille des caractères que vous souhaitez utiliser. Une boîte combo a les moyens d'afficher une liste des tailles disponibles et vous permettre de taper une autre valeur. Il serait possible d'employer pour cela un groupe de boutons radio, mais en cas d'une importante disponibilité de tailles, le groupe occupera sans doute trop de place dans la boîte de dialogue. Autre avantage, dans une boîte combo, le nombre des rubriques d'une liste peut varier au cours de l'exécution de l'application, alors que le nombre des boutons radio d'un groupe est généralement fixe. Pour ajouter un nouveau choix dans une boîte combo, il suffit au développeur d'insérer une chaîne de texte dans la boîte liste; en revanche, l'ajout d'une option supplémentaire dans un groupe de boutons radio exigerait la création d'un bouton de plus et la réorganisation du groupe tout entier.

Le premier élément d'une boîte combo est le champ d'édition monoligne. Un bouton placé à droite de ce dernier constitue le deuxième élément et le troisième se trouve dans une boîte liste située juste au-dessous du champ. Lorsque le contenu de la boîte liste défile, la sélection courante s'affiche dans le champ d'édition. La boîte liste d'une boîte combo possède les caractéristiques standard de celles de Windows 3.0, y compris les éléments owner-draw.

Windows 3.0 supporte trois styles de boîtes combos. Le style simple (le moins utilisé) dispose de l'indicateur CBS_SIMPLE. Dans une boîte combo simple, la boîte liste reste toujours visible. Ce genre de boîte ne possède aucun

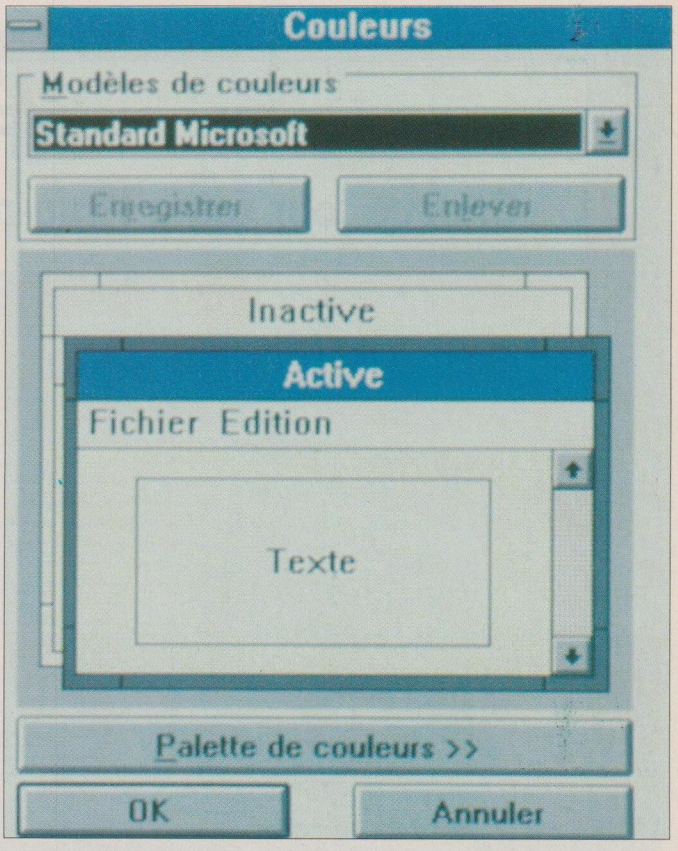


Figure 1. Le panneau de configuration de Windows 3.0 emploie une boîte combo «drop-down» pour la sélection des modèles de couleurs.

Boîtes combos, barres de défilement et contrôles définis par l'utilisateur : tels sont les thèmes traités dans ce second article consacré aux boîtes de dialogue et aux contrôles qui font partie intégrante de Microsoft Windows.

avantage par rapport à la définition d'un champ d'édition et d'une boîte liste séparée, sinon qu'il est inutile d'écrire le code qui régit la relation entre ces deux éléments.

Les deux autres types «évolués» de boîtes combos (*voir figu-re 1, photo ci-contre*) sont les «drop-down» et «drop-down list», possédant respectivement les styles CBS_DROP-DOWN et CBS_DROPDOWNLIST. La seule différence se trouve dans l'impossibilité pour les boîtes «drop-down list» d'autoriser la saisie dans le champ d'édition. Ce type est utile si vous souhaitez limiter le choix de l'utilisateur à une liste prédéfinie.

L'intérêt de ces deux types évolués est que la boîte liste reste cachée tant que vous ne la déroulez pas. Cette dernière manœuvre s'effectue soit en cliquant sur le bouton du champ d'édition, soit en pressant la combinaison Alt-Bas (touches Alt et flèche pointée vers le bas). Tant que le focus est affecté à la boîte, on peut employer les touches Haut, Bas, Page précédente et Page suivante pour faire défiler la liste.

Les messages et les codes de notification traités par le gestionnaire de messages d'une boîte combo sont comparables à ceux qu'utilisent les boîtes liste et les champs d'édition. La seule différence relève du préfixe CB_ ou CBN_ (au lieu des EM_, EN_, LB_ et LBN_). L'unique message particulier, CB_SHOWDROPDOWN, est employé pour afficher ou cacher la boîte liste (dans le cas des boîtes «drop-down» ou «drop-down list» seulement). Juste avant l'affichage de la liste, le code de notification CBN_DROPDOWN parvient à la fenêtre mère de la boîte combo. Ainsi, le programmeur a la possibilité de modifier le contenu de la liste avant de la présenter à l'utilisateur.

Les barres de défilement

De nombreuses applications Windows utilisent une fenêtre pour présenter (ou «découper») une portion d'un ensemble plus vaste. Une boîte liste est une fenêtre dans une suite de chaînes de texte. Une fenêtre d'édition d'un traitement de texte présente une portion d'un fichier, et ainsi de suite. Les barres de défilement constituent le moyen le plus simple

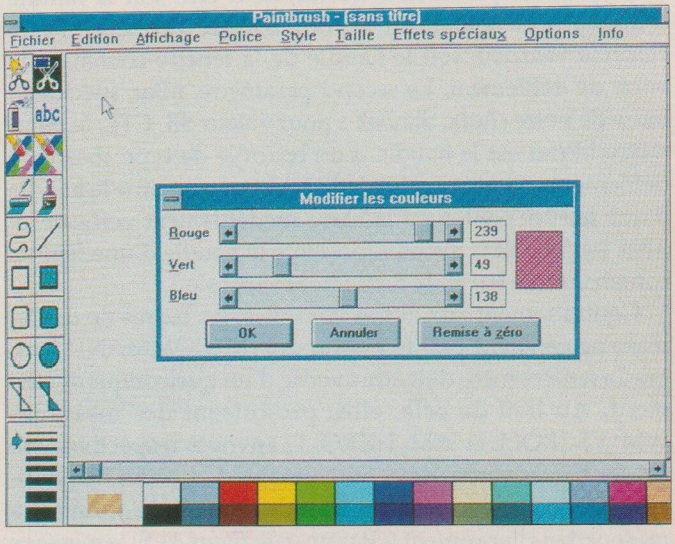


Figure 2. Des barres de défilement permettent de modifier les couleurs de PaintBrush.

d'informer l'application que l'utilisateur souhaite observer une autre partie de l'ensemble des données.

Une barre de défilement peut être verticale ou horizontale. Elle possède deux principaux attributs : l'intervalle des valeurs que représente la longueur de la barre et la position courante dans ce dernier. Celle-ci est indiquée par le petit icone carré appelé «curseur de défilement».

Les barres de défilement se trouvent généralement associées à une fenêtre au moment de la création de celle-ci. Pour adjoindre une barre verticale à une fenêtre, il suffit de donner à cette dernière le style WS_VSCROLL lors de son élaboration. De la même manière, le style WS_HSCROLL attache une barre de défilement horizontale au bas de la fenêtre.

Les barres de défilement peuvent aussi être employées comme des contrôles indépendants. Un bon exemple se situe dans la boîte de dialogue «Modifier les couleurs» de l'accessoire Windows PaintBrush (*voir figure 2, photo ci-dessus*).

Une barre de défilement indépendante est créée de la même manière que n'importe quel autre contrôle : en utilisant CreateWindow ou en définissant le contrôle comme la partie d'une boîte de dialogue dans un fichier ressource. Les barres de défilement verticales possèdent le style SBS_VERT,

les barres horizontales ont le style SBS_HORZ.

Les fonctions de l'API Windows autorisent la recherche et la définition de la position courante et de l'intervalle de visualisation. Les cinq fonctions des barres de défilement sont les suivantes :

```
int GetScrollPos(hwnd, nBar);
void GetScrollPos(hwnd, nBar, lpMin, lpMax);
int SetScrollPos(hWnd, nBar, iPos, bRedraw);
void SetScrollRange(hWnd, nBar, nMin, mMax,
bRedraw);
void ShowScrollBar(hWnd, nBar, bShow);
```

Le premier paramètre, hWnd, peut être soit le handle d'une barre de défilement (si l'on s'adresse directement à un contrôle «barre»), soit le handle de la fenêtre contenant la barre de défilement. Le second paramètre, nBar, spécifie la barre de votre choix. Si nBar a pour valeur SB_CTL, le paramètre hWnd est le handle d'un contrôle de type «barre de défilement». Si nBar est SB_VERT, hWnd constitue le handle d'une fenêtre contenant la barre de défilement verticale; si nBar est SB_ORZ, hWnd représente le handle d'une fenêtre contenant une barre de défilement verticale.

Contrairement aux autres contrôles, les barres de défilement ne génèrent pas de messages WM_COMMAND lorsque la fenêtre mère doit être avertie d'un quelconque événement. Au lieu de cela, elles produisent des messages WM_VSCROLL et WM_HCROLL, envoyés respectivement par les barres verticales et horizontales. Les codes de notification de ces messages sont les mêmes, et ils se trouvent à présent dans l'argument wParam (*voir figure 3*). Si le message a été généré par un contrôle de type barre de défilement, le handle de la barre est passé dans le mot de poids fort (HIWORD) de l'argument lParam (sinon le HIWORD de lParam est à 0). Le mot de poids faible (LOWORD) de lParam contient la position courante du curseur de défilement pour ces deux messages de notification.

Figure 3 : Codes de notification dans les messages des barres de défilement.

Valeur de wParam	Signification
SB_BOTTOM	Déplacement vers le bas/la droite -- utilisé uniquement avec les contrôles "barres de défilement"
SB_ENDSCROLL	Fin du défilement
SB_LINEDOWN	Déplacement d'une ligne vers le bas/Déplacement vers la droite
SB_LINEUP	Déplacement d'une ligne vers le haut/Déplacement vers la gauche
SB_PAGEDOWN	Déplacement d'une page vers le bas/Déplacement vers la droite
SB_PAGEUP	Déplacement d'une page vers le haut/Déplacement vers la gauche
SB_THUMBPOSITION	Déplacement vers une position absolue -- la position courante est fournie par le mot de poids faible de lParam
SB_THUMBTRACK	Tire le curseur de défilement vers la position spécifiée -- la position courante est fournie par le mot de poids faible de lParam
SB_TOP	Déplacement vers le haut/la gauche -- utilisé uniquement avec les contrôles "barres de défilement"

Lorsque l'on clique sur la flèche supérieure d'une barre de défilement verticale ou sur la flèche gauche d'une barre de défilement horizontale, le code de notification SB_LINEU est envoyé à la fenêtre mère. En cliquant sur la flèche inférieure ou sur la flèche droite, cela génère un code SB_LINEDOWN. Le fait de cliquer dans la zone située entre le sommet (ou le côté gauche de la barre) et le curseur de défilement déclenche l'envoi d'un message SB_PAGEUP. Enfin, si l'on clique entre le bas (ou le côté droit) de la barre et le curseur de défilement, un message SB_PAGEDOWN est engendré. Quand on déplace le curseur de défilement, la barre produit une suite de codes de notification SB_THUMBTRACK en plaçant la position du curseur de défilement dans le LOWORD de lParam. Les codes SB_THUMBTRACK permettent au développeur de modifier le contenu de la fenêtre à mesure que le curseur se trouve déplacé. Néanmoins, il n'est pas toujours possible de mettre à jour «simultanément» le contenu d'une fenêtre (cela arrive souvent dans le cas d'un contenu de fenêtre trop complexe). Quand le curseur de défilement s'arrête, la barre génère le message SB_THUMBPOSITION, qui contient la position finale du curseur dans le LOWORD de lParam. Si le défilement dynamique du contenu de la fenêtre se révèle impossible, cette dernière peut être mise à jour seulement lorsque le déplacement du curseur est achevé.

Une fois ces codes de notification envoyés à la WinProc de la fenêtre mère d'une barre de défilement, cette procédure

de fenêtre doit actualiser d'une part le curseur de défilement pour indiquer la nouvelle position, d'autre part, les données contenues dans la fenêtre. Windows gère automatiquement le défilement des boîtes liste, des boîtes combos et des champs d'édition dotés de barres de défilement.

Exemple d'utilisation d'une barre de défilement

Supposons que vous ayez écrit une application Windows simple produisant constamment un son (tels les programmes destinés aux guitaristes pour accorder leur instrument). Vous attachez une barre de défilement verticale à la fenêtre principale du programme; l'intervalle de défilement de la barre représente huit octaves (une octave comprenant douze notes, cela fait un total de 96 notes) et la position courante du curseur de défilement correspond à la note produite par le haut-parleur (vous disposez, par exemple, d'une fonction SoundTone, qui accepte la valeur d'une note et joue le son correspondant). Commençons par créer une fenêtre possédant une barre de défilement et spécifions un intervalle de défilement de 1 à 96. Le son initial est défini comme constituant un La de 440 Hz — la 22e note de notre intervalle (*voir figure 4*).

Pour obtenir la note précedant ou suivant l'intervalle, il faut cliquer respectivement sur la flèche supérieure ou sur la flèche inférieure de la barre de défilement. Pour passer à l'octave supérieure ou inférieure, l'utilisateur clique entre le curseur de défilement et le sommet ou la base de la barre (qui doit être suffisamment grande). Le déplacement du curseur le long de la barre produit un effet de glissando (*voir figure 5*). Les messages WM_VSCROLL envoyés à la fenêtre principale nécessitent un traitement afin de produire le son voulu. Il faut également actualiser le curseur de défilement afin de le placer dans la nouvelle position correcte sur la barre de défilement.

Figure 4 : Création et initialisation d'une barre de défilement.

```
hWndMain = CreateWindow("Accorder",
                        "Application Accord",
                        WS_OVERLAPPEDWINDOW |
                        WS_VSCROLL,
                        CW_USEDEFAULT, 0,
                        CW_USEDEFAULT, 0,
                        (HWND) NULL,
                        (HMENU) NULL,
                        hThisInstance,
                        (LPSTR) NULL);

SetScrollRange(hWndMain,
               SB_VERT,1,96,FALSE);

SetScrollPos(hWndMain,
             SB_VERT,iCurrTone=22,FALSE);

ShowWindow(hWnd, nCmdShow);
UpDateWindow(hWnd);
SoundTone(iCurrTone);
```

Figure 5 : Traitement des messages de la barre de défilement.

```
case WM_VSCROLL :
  switch (wParam)
  {
    case SB_LINEUP :
      iCurrTone = max(1, iCurrTone - 1);
      break; /* Baisser d'un demi-ton */
    case SB_LINEDOWN :
      iCurrTone = max(96,iCurrTone + 1);
      break; /* Monter d'un demi-ton */
    case SB_PAGEUP :
      iCurrTone = max(1,iCurrTone - 12);
      break; /* Baisser d'une octave */
    case SB_PAGEDOWN :
      iCurrTone = max(96,iCurrTone +12);
      break; /* Monter d'une octave */
    case SB_THUMBTRACK :
    case SB_THUMBPOSTION :
      iCurrTone = LOWORD(lParam);
      break; /* Glissando */
  }
  SoundTone(iCurrTone);
  SetScrollPos(hWnd,SB_VERT,iCurrTone,
  TRUE);
  break;
```

Les boîtes de dialogue

On peut symboliser les boîtes de dialogue comme la «colle» qui réunit les groupes de contrôles. Elles ne sont que des fenêtres pop-up contenant une ou plusieurs fenêtres filles. Ces dernières peuvent appartenir à l'une des classes de contrôle que nous avons examinées, ou bien à une classe développée par le programmeur. Les boîtes de dialogue se distinguent des fenêtres pop-up ordinaires par deux caractéristiques. D'une part, une boîte de dialogue peut être définie dans un fichier ressources Ascii externe. D'autre part, dans Windows, un gestionnaire de boîte de dialogue régit l'interaction entre l'utilisateur et la boîte.

Ce gestionnaire traite les frappes des touches du clavier et accomplit les actions correspondantes. Si l'utilisateur presse la touche de tabulation, le gestionnaire doit chercher le prochain contrôle qui peut recevoir le focus dans la boîte (autrement dit, la vérification suivante possédant le style WS_TABSTOP). Lorsque la touche Esc est enfoncée, le gestionnaire génère un message WM_COMMAND ayant pour valeur IDCANCEL ainsi que l'envoi à la procédure de la boîte.

Plusieurs étapes sont nécessaires pour pouvoir utiliser une boîte de dialogue dans une application Windows. En

premier lieu, il faut définir celle-là dans le fichier ressources (.RC), compiler ce dernier à l'aide du compilateur de ressources et l'attacher au fichier.EXE de votre application. Ensuite, celle-ci doit définir une procédure de boîte de dialogue afin de traiter les messages transmis à cette boîte. Enfin, il faut que votre application charge la boîte de dialogue à partir du fichier ressources et appelle le gestionnaire de boîte de dialogue afin d'afficher cette dernière — et faire en sorte qu'il puisse y avoir une interaction avec l'utilisateur.

Il existe deux types de boîtes de dialogue, modal et non modal. La plupart des applications utilisent des boîtes modales afin de recueillir des informations fournies par l'utilisateur. Tant qu'elles n'auront pas été refermées, il vous sera impossible de vous adresser ou de passer à une autre fenêtre de votre application (autrement dit, tant que vous n'aurez pas mis fin au «dialogue»). En fait, lorsque vous appelez une boîte de dialogue modale, son gestionnaire entre dans sa propre boucle de messages et ne répartit que ceux destinés à la boîte ou à l'un de ses contrôles. En règle générale, l'utilisateur arrête le dialogue en cliquant sur un bouton poussoir «OK» ou «Annuler», ou en pressant les touches Esc ou Entrée (Enter). Les boîtes de dialogue non modales vous autorisent à leur retirer le focus pour le donner à un autre composant de l'application. Vous déplacez celui-là en cliquant sur une autre fenêtre. L'éditeur de fichiers PIF de Windows en constitue un bon exemple. Avec PIFEdit, vous pouvez activer le menu déroulant alors que le focus se trouve affecté à l'une des boîtes de dialogue non modales. Comme les méthodes de chargement et d'utilisation des boîtes modales et non modales sont très différentes, nous examinerons chacune d'elles.

Définir une boîte de dialogue dans un fichier ressource

Un programmeur Windows expérimenté évite de définir «manuellement» une boîte de dialogue et ses contrôles dans un fichier ressources, car il est beaucoup plus simple et pratique d'utiliser le Windows Dialog Editor (*voir figure 6, photo ci-contre*) fourni avec le kit de développement de Windows («Microsoft Windows Software Development Kit», communément appelé SDK). De nombreuses options et beaucoup de styles existent pour les boîtes de dialogue et les fenêtres de contrôles. Le Dialog Editor vous permet de dessiner et de construire une boîte de dialogue de manière interactive et de produire automatiquement la définition nécessaire au fichier de ressources.

La figure 7 montre la syntaxe employée pour la définition d'un exemple de boîte de dialogue. Les premières lignes spécifient le nom, les options de chargement, les options mémoire et les coordonnées de la boîte. Par défaut, les deux types d'options sont LOADONCALL et MOVEABLE. Une ressource LOADONCALL ne se trouve chargée en mémoire qu'au moment où l'application en a besoin. Cela représente l'un des principaux avantages des fichiers ressources.

Les coordonnées spécifiées dans les définitions des boîtes de dialogue s'emploient d'une manière particulière. Comme les applications doivent être le plus possible indépendantes du contexte, il faut que les boîtes de dialogue aient toujours le même aspect, quel que soit le type d'adaptateur vidéo utilisé. Si une boîte est censée occuper la totalité de la surface d'affichage, elle doit le faire aussi bien pour une résolution de 640 x 350 que pour une autre de 1 024 x 768. Pour cela, le système de coordonnées de la boîte utilise «l'unité de base de dialogue». Il s'agit d'une unité de mesure calculée en fonction de la police système courante. Dans une définition de boîte, la coordonnée x et la largeur sont données en 1/8 d'unité de base de dialogue; la coordonnée y et la hauteur s'expriment en 1/4 d'unité de base de dialogue. Pour déterminer le nombre exact de pixels dans les unités de base horizontale et verticale, on peut employer une nouvelle fonction de Windows 3.0 : GetDialogBaseUnits.

Après avoir défini le nom, les options de chargement, les options mémoire et les coordonnées de la boîte de dialogue, il est possible de déterminer d'autres options. On peut employer l'instruction Style pour donner certains attributs de style à la boîte — les mêmes que ceux fournis à une fonction CreateWindow. Quatre styles sont caractéristiques des boîtes de dialogue : DS_LOCALEDIT, DS_MODALFRAME, DS_NOIDLEMSG et DS_SYSMODAL. Ce dernier crée une boîte de dialogue modale, toutes les fenêtres du système étant désactivées aussi longtemps que l'on s'y trouve. Ce style de boîte devrait être employé, par exemple, pour signaler une erreur fatale qui risquerait d'affecter la poursuite de la session Windows.

On peut donner un titre à la boîte grâce à l'instruction

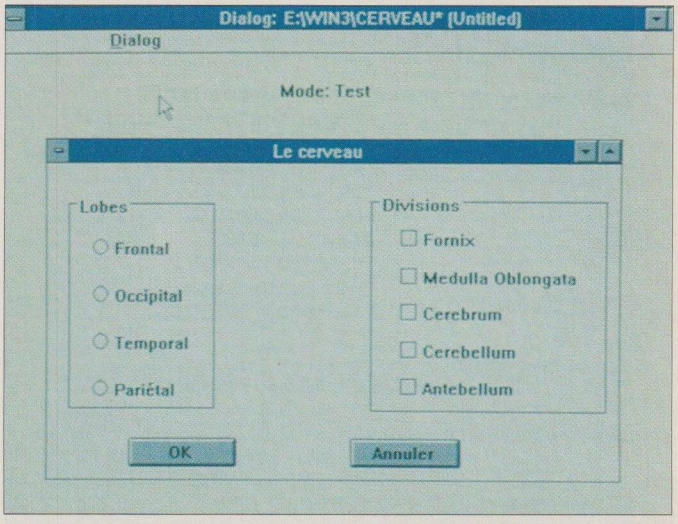


Figure 6. Création d'une boite de dialogue avec le Dialog Editor.

Figure 7 : Définition de la boîte de dialogue " Ajouter une cotation".

```
ADDTICK DIALOG LOADONCALL MOVEABLE
DISCARDABLE 112, 31, 106, 86
  CAPTION "Ajouter une cotation"
  STYLE WS_BORDER | WS_CAPTION |
WS_DLGFRAME | WS_POPUP
BEGIN
    CONTROL "Date:", -1, "static",
SS_LEFT | WS_CHILD, 2, 7, 22, 8
    CONTROL "", ID_TICK_DATE, "edit",
ES_LEFT | WS_BORDER | WS_TABSTOP |
WS_CHILD, 53, 5, 48, 12
    CONTROL "Prix de cl_ture:", -1,
"static", SS_LEFT | WS_CHILD, 2, 26, 55,
11
    CONTROL "", ID_TICK_PRICE, "edit",
ES_LEFT | WS_BORDER | WS_TABSTOP |
WS_CHILD, 57, 25, 44, 12
    CONTROL "Volume:", -1, "static",
SS_LEFT | WS_CHILD, 2, 44, 32, 8
    CONTROL "", ID_TICK_VOLUME, "edit",
ES_LEFT | WS_BORDER | WS_TABSTOP |
WS_CHILD, 38, 43, 63, 12
    CONTROL "OK", 1, "button",
BS_DEFPUSHBUTTON | WS_TABSTOP | WS_CHILD,
9, 66, 28, 14
    CONTROL "Annuler", 2, "button",
BS_PUSHBUTTON | WS_TABSTOP | WS_CHILD,
63, 66, 32, 14
END
```

Caption. Les autres options sont définies par les instructions Menu, Class et Font. Celle-ci, introduite dans la version 3.0 de Windows, permet de spécifier le type et la taille des caractères utilisés dans la boîte de dialogue.

Lorsque cette information d'en-tête a été spécifiée dans la définition de la ressource, il suffit de définir une liste des contrôles, délimitée par les instructions Begin et End. Mais le Dialog Editor du SDK accomplit tout cela pour vous; il vous permet de placer graphiquement les contrôles et se charge d'écrire les définitions du fichier RC.

La définition de la boîte de dialogue «Ajouter une cotation» de notre exemple d'application apparaît dans la figure 7 (*voir aussi figure 18*). Cette définition a été produite par le Dialog Editor. Généralement, il n'y a guères de difficultés à savoir si une définition de boîte de dialogue a pour origine le Dialog Editor, car celui-ci produit les formes longues des instructions Control.

Les arrêts de tabulation et les groupes

Nous avons dit plus haut que le gestionnaire de boîte de dialogue gère les frappes de touches destinées à la boîte et que, lors d'une pression sur la touche de tabulation, il déplace le focus vers le contrôle suivant. Comment peut-il connaître le contrôle auquel attribuer le focus ?

Le bit de style WS_TABSTOP doit être mis à 1 pour chaque contrôle vers lequel on peut se déplacer à l'aide de la tabulation avant ou arrière. C'est particulièrement important pour ceux définis par le programmeur, car le gestionnaire de boîte de dialogue n'a aucun autre moyen de savoir si un contrôle «personnalisé» doit simplement être affiché ou s'il nécessite une interaction avec l'utilisateur (si un contrôle statique possède le style WS_TABSTOP, le gestionnaire doit rechercher le prochain contrôle non statique qui le suit, afin de déplacer le focus d'entrée vers un contrôle valide).

Le gestionnaire de boîte de dialogue supporte encore ce que l'on nomme «un groupe contrôle». Dans ce cas, les contrôles sont organisés de manière à ce que l'on puisse se déplacer parmi eux à l'aide des touches fléchées plutôt qu'avec celle de tabulation. L'exemple le plus commun est une boîte de groupe contenant des boutons radio. Si le premier bouton radio du groupe possède le style WS_GROUP et qu'il partage avec le premier contrôle défini en dehors du groupe, vous pourrez vous déplacer parmi les boutons radio du groupe à l'aide des touches fléchées. Dans ce cas, on utilise alors la touche de tabulation pour passer à un contrôle extérieur au groupe radio. En général, pour créer un groupe contrôle, on donne le style WS_GROUP au premier contrôle du groupe et au premier contrôle défini ensuite hors du groupe.

Si vous désirez obtenir le handle de fenêtre du prochain élément dans un groupe contrôle, vous pouvez employer la fonction GetNextDlgGroupItem. Celle-ci s'utilise pour personnaliser le traitement des séquences de touches dans une boîte de dialogue. Par exemple, si une application Windows permet de personnaliser le clavier, l'utilisateur a la possibilité de définir une combinaison de touches qui obtiendra le même effet que la flèche Bas (c'est le cas de Ctrl-X avec WordStar). Quand l'utilisateur presse Ctrl-X dans une boîte de dialogue, le focus doit être déplacé vers le prochain membre du groupe contrôle. On peut employer la fonction GetNextDlgGroupItem pour recevoir le handle du prochain contrôle afin de lui donner le focus. Une fonction similaire, GetNextDlgTabItem, s'emploie pour les contrôles de style WS_TABSTOP.

Charger une boîte de dialogue

Maintenant que la boîte de dialogue a été définie, compilée et attachée au fichier EXE, il faut la disposer dans l'application. Quatre fonctions permettent de charger ou de créer une boîte de dialogue modale et de la gérer à l'aide du gestionnaire correspondant. Deux de ces fonctions, DialogBox et DialogBoxParam, chargent donc la boîte à partir du fichier RES attaché au fichier EXE. Les autres fonctions, DialogBoxIndirect et DialogBoxIndirectParam, créent des boîtes de dialogue à l'aide d'une description modèle stockée dans l'ap-

plication. Comme cette dernière méthode reste inhabituelle, nous ne la décrirons pas ici.

La fonction employée pour charger et invoquer une boîte de dialogue est DialogBox :

```
   FARPROC lpfn;
   lpfn = MakeProcInstance((FARPROC) OpenDlg,
hThisInstance);
   DialogBox(hThisInstance, "Ouvrir", hWnd,
 lpfn);
   FreeProcInstance(lpfn);
```

L'appel à MakeProcInstance récupère l'adresse de la procédure de la boîte de dialogue (procédure écrite par le programmeur) et crée une petite portion de code appelé «thunk». Un thunk commence par lier les données spécifiées par hThisInstance à la fonction pointée par OpenDlg, puis effectue un branchement vers cette dernière. MakeProcInstance retourne un identificateur long (un pointeur) fournissant l'adresse du thunk. Lorsque l'invocation de la boîte de dialogue s'achève, il faut libérer le thunk en appelant FreeProcInstance (les thunks sont des mécanismes complexes et qui se trouvent décrits en détail dans «Programming Windows», Microsoft Press, 1990).

La fonction DialogBox possède quatre arguments : 1) le handle de l'instance de l'application (ne pas oublier qu'il s'agit d'un identificateur de programme possédant une valeur unique, ce qui permet à Windows de distinguer plusieurs copies d'une même application qui s'exécutent simultanément); 2) le nom de la boîte de dialogue à charger; 3) le handle de la fenêtre propriétaire; 4) le pointeur vers le thunk de l'instance. La valeur retournée par DialogBox correspond à celle que retourne la procédure de boîte de dialogue écrite par le programmeur (en fait, la valeur de retour devient celle qui est passée comme second argument à la fonction EndDialog).

Procédure de boîte de dialogue

Une procédure, définie par le programmeur, doit se trouver associée à chaque boîte de dialogue. On peut comparer la procédure de boîte de dialogue à une WinProc standard, excepté quelques distinctions. Tout d'abord, une procédure de boîte de dialogue retourne une valeur booléenne, True ou False (Vrai ou Faux). A la différence d'une WinProc standard, votre procédure de boîte de dialogue doit renvoyer True si elle traite un message et False dans le cas contraire.

Quand on invoque une boîte de dialogue modale, le gestionnaire de boîte de dialogue entre dans sa propre boucle de messages internes. Chaque message destiné à la boîte passe d'abord celle-ci à la procédure associée. Si cette procédure retourne la valeur False, le gestionnaire passe alors le message à sa propre procédure par défaut. Si la valeur True a été retournée, le gestionnaire ne s'occupe plus du message et lit le suivant. Windows 3.0 permet au programmeur expérimenté d'avoir accès à la procédure de boîte de dialogue par défaut, DefDlgProc. Microsoft distribue le code source de DefDlgProc avec le SDK. La procédure de boîte de dialogue doit être définie comme suit :

```
BOOL FAR PASCAL MyDialogProc (hDlg,mes-
 sage,wParam, lParam)
   HWND    hlg;
   WORD    message;
   WORD    wParam;
   DWORD   lParam;
```

Aucune différence ne distingue cet en-tête de celui d'une WinProc, à part la valeur de retour de type Bool. Et, comme une WinProc, la procédure de boîte de dialogue doit être exportée dans le fichier DEF de votre application :

```
EXPORTS
    .
    .
    .
MyDialogProc
```

Si le comportement d'une boîte de dialogue semble mauvais, vérifiez que vous avez bien exporté la fonction associée à la boîte de dialogue. Cet oubli constitue, en effet, une erreur courante chez les programmeurs Windows novices (et même chez les plus expérimentés).

Les plus simples des procédures de boîte de dialogue ne traitent que les messages WM_INITDIALOG et WM_COMMAND. WM_INITDIALOG, semblable au message WM_CREATE, est envoyé par le gestionnaire de boîte de dialogue juste avant le premier affichage de la boîte. Il offre à la procédure de la boîte la possibilité d'initialiser le contenu des contrôles de la boîte. Par exemple, la procédure peut remplir le contenu du champ d'édition avec une valeur par défaut, placer des chaînes dans la boîte liste et vérifier l'état de certains boutons radio ou de quelques cases à cocher. Si l'on ne souhaite pas que le gestionnaire de boîte de dialogue attribue automatiquement le focus initial au premier contrôle ayant le style WS_TABSTOP, le focus peut être transmis à un autre contrôle. S'il devait changer, la procédure de la boîte de dialogue retournerait False après avoir traité le message WM_INITDIALOG. Si elle renvoie la valeur True, le gestionnaire de boîte de dialogue affectera automatiquement le focus (ce message représente l'exception à la règle précisée plus haut à propos des valeurs booléennes retournées par la procédure de la boîte de dialogue).

L'élément lParam du message WM_INITDIALOG n'était pas employé par les versions précédentes de Windows. Désormais, la version 3.0 permet de passer une valeur longue au gestionnaire de boîte de dialogue, qui la transmettra ensuite à la procédure de la boîte de dialogue — comme argument lParam du message WM_INITDIALOG. Pour cela,

nous employons DialogBoxParam, une fonction quasi identique à DialogBox, mais à laquelle un cinquième argument s'ajoute : la valeur longue à passer dans le message WM_INITDIALOG.

La procédure de la boîte de dialogue ne se contente pas de cette passation, elle se charge surtout du traitement des messages WM_COMMAND générés lorsque l'utilisateur manipule un contrôle. Ils se présentent habituellement sous deux formes : ceux produits quand on presse un bouton et ceux envoyés, avec les codes de notification, dans le mot de poids fort de lParam.

Lorsque l'utilisateur presse sur OK ou Annuler, cela signifie habituellement qu'il souhaite fermer la boîte de dialogue. Un message WM_COMMAND est transmis à la procédure de la boîte et l'élément wParam de ce message contient l'identificateur du bouton poussoir. Grâce à un appel à EndDialog, le gestionnaire de boîte de dialogue est informé que l'utilisateur a fini d'utiliser la boîte.

EndDialog possède deux arguments : le handle de la boîte de dialogue et une valeur sur un mot. Cette dernière sera employée comme valeur de retour de l'appel à DialogBox :

```
switch (message)
{
   case WM_COMMAND :
   switch (wParam)
{
      case IDOK :
EndDialog (hDlg, TRUE);
   break;
      break;
      case IDCANCEL :
EndDialog (hDlg, FALSE);
      break;
```

Si l'utilisateur a pressé OK, EndDialog est appelée pour signifier au gestionnaire de boîte de dialogue la fermeture de celle-ci. Il retournera la valeur True à l'application, et dans l'instruction suivante, la renverra et l'affectera à la variable rc.

```
   rc = DialogBox(hInstance, "Ajouter",
hWndMain,
          lpfnDialogProc);
```

De la même manière, si l'utilisateur clique sur Annuler, la valeur False sera retournée à l'application.

EndDialog ne ferme pas la boîte immédiatement. Elle valide un bit dans une structure de données privée allouée à chaque boîte de dialogue par le gestionnaire (elle copie également la valeur du second argument dans cette structure). Quand la procédure de votre boîte rend la main au gestionnaire de boîte de dialogue, celui-ci examine le bit en question et, s'il est à 1, sort de sa boucle de messages.

Les boîtes de message

En règle générale, les programmes Windows informent l'utilisateur d'une condition d'erreur potentielle ou réelle. Par exemple, si l'utilisateur tente de quitter un traitement de texte sans avoir sauvegardé le document, l'application affiche habituellement une boîte de dialogue contenant un message d'avertissement et un choix d'actions à entreprendre. Le message ressemble normalement à ceci : «*Voulez-vous sauver les modifications avant de quitter l'application ?*»; trois boutons Oui, Non et Annuler, l'accompagnent.

Il serait très pénible pour le programmeur d'avoir à construire des boîtes de dialogue pour chaque condition d'erreur ou message d'avertissement. Heureusement, on peut appeler une unique fonction affichant un texte dans une boîte de dialogue en y associant un ou plusieurs boutons prédéfinis. Ce genre de boîtes est appelé «boîte de message»; la fonction employée pour les créer et les afficher s'intitule MessageBox.

Windows contient la procédure de boîte de dialogue chargée de toutes les boîtes de message et gère tous les messages WM_COMMAND. Cela évite de créer votre propre procédure de boîte de dialogue, de l'exporter et de la charger. Il suffit de traiter le code renvoyé par MessageBox.

Voici l'en-tête de la fonction MessageBox :

```
 int MessageBox(HWND hWndParent, LPSTR
lpText,LPSTR lpCaption, WORD wType)
```

Le premier paramètre est le handle de la fenêtre propriétaire. Le texte statique qui apparaît dans la boîte de message constitue le deuxième. En cas de longueur excessive pour tenir sur une seule ligne, Windows l'affiche automatiquement sur plusieurs et adapte la hauteur de la boîte. Le troisième paramètre est un texte facultatif placé dans le titre de la boîte. Des drapeaux binaires précisant à Windows quels types de boutons ou d'icones il faut placer dans la boîte de message représentent le dernier paramètre. Si vous désirez un bouton poussoir différent de ceux prédéfinis, vous devez définir vous-même la boîte de message. Windows vous propose les boutons poussoirs Oui, Non, Annuler, Abandon, Ignorer et Répéter. Il vous permet également de placer l'une des icones système prédéfinis dans la partie gauche de la boîte de message (*voir figure 8, photo ci-dessous*).

La valeur retournée par MessageBox est l'identificateur de contrôle du bouton poussoir sélectionné par l'utilisateur (si l'utilisateur presse OK, la valeur IDOK se trouve ren-

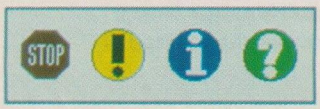


Figure 8. Windows propose des icones prédéfinis pour la fonction Message Box.

voyés; en appuyant sur Non, IDNO est retournée; et ainsi de suite). Le fichier WINDOWS.H donne la liste complète des identificateurs de ces boutons (*voir figure 9*). Voici une utilisation typique de la boîte de message :

```
rc =MessageBox(hWndMain, "Voulez-vous quitter l'application?", "Quitter", MB_YESNO |
MB_ICONQUESTION);
    if (rc == IDYES)
       PostQuitMessage(0);
```

Figure 9 : Identificateurs de boutons dans le fichier WINDOWS.H.

```
/* Identificateurs de commande pour une boîte de dialogue */

#define IDOK            1
#define IDCANCEL        2
#define IDABORT         3
#define IDRETRY         4
#define IDIGNORE        5
#define IDYES           6
#define IDNO            7
```

Retour aux boîtes de dialogue non modales

Les boîtes de dialogue non modales sont comparables à des hybrides de fenêtres normales «overlapped» et de boîtes de dialogue modales. Pour créer l'une de ces dernières, utilisez CreateDialog. La syntaxe de cette fonction est identique à celle de DialogBox. Cependant, lorsque l'on conçoit une boîte non modale, la fonction CreateDialog effectue un retour immédiat en fournissant le handle de la boîte de dialogue élaborée; Windows n'invoque pas le gestionnaire interne de boîte de dialogue. Voici un appel caractéristique permettant de composer une boîte de dialogue :

```
hDlgModeless = CreateDialog(hInstance, "DlgNonModale",
        hWndMain, lpfnProc);
```

Comme une boîte non modale ne constitue qu'une fenêtre «overlapped» standard ayant des fenêtres filles, des messages lui sont envoyés comme à toute autre fenêtre de l'application. Il suffit pour cela d'utiliser la boucle de messages normale de l'application:

```
while (GetMessage(&msg, 0, 0, 0))
{
   TranslateMessage(&msg);
   DispatchMessage(&msg);
}
```

Mais il manque alors quelque chose. Supposons que le focus d'entrée se situe sur l'un des contrôles de la boîte non modale lorsque vous pressez la touche de tabulation. Il ne se passe rien, car le gestionnaire de boîte de dialogue n'est pas là pour traiter les frappes de touches du clavier et donner une signification particulière à la touche de tabulation. Bien entendu, vous pourriez écrire dans votre application le code de traitement de la touche, mais faire appel temporairement au gestionnaire de boîte de dialogue constitue une solution beaucoup plus simple.

Windows remédie à ce problème. La fonction IsDialogMessage(hDlg, &msg) exécute un test pour déterminer si le message contenu dans la structure MSG est destiné à la boîte de dialogue dont le handle se trouve dans le premier argument. Une fois cette condition remplie, IsDialogMessage effectue toutes les conversions de touches nécessaires et envoie le message lui-même à la boîte de dialogue. La fonction se charge d'interpréter les frappes, puis elle retourne True si le message a été traité, et False dans le cas contraire. La boucle de messages standard, modifiée pour inclure l'emploi de IsDialogMessage, se présente comme suit :

```
while (GetMessage(&msg, 0, 0, 0)
{
   if ( !hDlgModeles | |
   !IsDialogMessage(hDlgModeless, &msg))
   {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }
}
```

Une fois le message obtenu, il passe à la fonction IsDialogMessage. Si la valeur True est renvoyée, vous savez qu'elle a traité complètement le message et qu'il devient inutile d'appeler TranslateMessage et DispatchMessage.

Toutefois, il faut considérer un dernier aspect des boîtes de dialogue non modales, c'est que le gestionnaire de dialogue reste inemployé, et qu'on ne peut donc pas se servir de la fonction EndDialog pour achever le traitement et détruire la boîte. Au lieu de cela, on doit procéder à un appel standard à DestroyWindow. Il faut encore invalider le handle de la boîte de dialogue pour que la fonction IsDialogMessage ne soit plus appelée dans la boucle de messages.

```
DestroyWindow(hDlg);
hDlgModeless = 0;
```

L'exemple d'application

A la fin de l'article précédent, notre application de portefeuille d'actions possédait un système complet de menu déroulant et des capacités MDI (documents multiples). A chaque fois que l'on choisissait la rubrique Fichier/Ouvrir, une nouvelle fenêtre fille MDI était créée. Les rubriques du menu Fenêtre se chargeaient de placer les fenêtres filles en mosaï-

que ou en cascade et de réorganiser celles réduites en icone. Mais l'application ne faisait que cela; aucune partie du code ne concernait les actions. Maintenant que l'on a compris le fonctionnement des boîtes de dialogue, nous allons implémenter un formulaire d'entrée pour demander à l'utilisateur des informations sur les actions. Il faudra déterminer quelles sortes de renseignements seront associées à chaque action, et créer une structure de données décrivant un objet «action». Le stockage de l'information relative à l'action devra permettre également un accès facilité. Comme cette application n'est pas destinée à une utilisation commerciale (et comme il s'agit ici d'apprendre Windows, et non les structures de données), nous ne nous préoccuperons pas des performances mais plutôt de faciliter l'implémentation et la compréhension. Nous aborderons éventuellement les quelques moyens d'obtenir des résultats plus satisfaisants.

Les modules nécessaires à la création du fichier STOCK.EXE sont décrits dans la figure 10. Il ne s'agit pas d'un système de gestion de portefeuille d'actions en temps réel : l'application ne gère pas le comportement de l'action tout au long de la période d'ouverture de la Bourse. Nous ne nous occuperons que de la valeur de clôture, et éventuellement du volume. D'autres types d'analyses pourraient prendre en compte la valeur moyenne ou les valeurs maximale et minimale, mais nous laisserons cela au lecteur, en guise d'exercice.

On peut encore souhaiter garder des informations sur les dates. Voici une structure permettant de conserver la datation :

```
typedef struct tagDate
{
   BYTE chMonth; /* Mois */
   BYTE chDay; /* Jour */
   BYTE chYear;/* Année */
} DATE;
```

Figure 10 : Le portefeuille d'actions, fichier STOCK.EXE.

```
STOCK.H
STOCK.RC
DB.C
FILEOPEN.C
STOCK
STOCK.DEF
STOCK.C
GRAPH.C
TICK.C
```

Figure 11 : Structure de données des graphes.

```
/*
Structure décrivant l'affichage du graphe
*/
typedef struct tagGraphInfo
{
  DWORD dwMinPrice;
  DWORD dwMaxPrice;
  DWORD dwScaleFactor;
  DWORD dwTickInterval;
  WORD  iDenominator; /* fraction
employée pour cette action */
  WORD  iGridPen;
} GRAPHINFO, FAR *LPGRAPHINFO;
```

Six octets sont utilisés pour chaque date; on peut cependant en supprimer deux si l'on stocke la date à partir du 1er janvier 1900, dans un entier non signé. Au lieu de conserver la date avec chaque cotation, certains programmes d'analyse ne gardent qu'une suite de prix et de volumes.

Pour éviter la perte d'informations due à l'arithmétique en virgule flottante, on peut stocker le prix sous la forme d'une valeur de quatre octets à l'aide de cette formule :

prix = (montant en dollars) X (dénominateur de fraction) + (fraction)

Par exemple, un prix de 16 7/8 serait stocké sous la forme (16 * 8) + 7 = 135.

On peut maintenant définir une structure contenant des informations sur les échanges quotidiens :

```
typedef DWORD PRICE;
typedef DWORD VOLUME;
typedef struct tagTick
{
   PRICE price;
   VOLUME dwVolume;
   DATE date;
} TICK, FAR *LPTICK;
```

Une structure de données décrivant les propriétés du graphe de l'action est également nécessaire (*voir figure 11*). Ce sujet se verra abordé dans un prochain article où nous construirons la structure adéquate.

Toutes ces informations doivent pouvoir être transférées et récupérées sur disque, sans quoi le programme perdrait de son utilité. En plus des de cotation, il faut sauvegarder le nom de l'action (un symbole de cinq lettres au maximum), sa description , les paramètres d'affichage du graphe et le nombre de cotations. Placer une signature au début du fichier permet aussi de s'assurer que celui lu sur disque correspond effectivement à un fichier d'action (*voir figure 12*).

Enfin, il nous faut une structure pour conserver toutes les informations concernant une action pendant l'exécution de l'application. En plus de l'ensemble des renseignements

Figure 12 : Etiquette d'un fichier d'actions

```
#define MAXSTOCKNAME      5
#define MAXDESCRIPTION    32
#define MAXFILENAME       13

typedef struct tagStockFile
{
  DWORD  dwMagic;
#define MAGIC_COOKIE  66666666L
  char   szStock[MAXSTOCKNAME];
  char   szDescription[MAXDESCRIPTION];
  GRAPHINFO graphinfo;
  WORD   nTicks;
/*
  TICK   aTicks[1];
*/
} STOCKFILE;
```

contenus dans le fichier d'action, nous devons stocker le nom de ce dernier, le handle de la fenêtre fille MDI dans laquelle l'information est affichée, un handle sur un tampon mémoire contenant les données de cotation et quelques drapeaux déterminant l'état courant de l'action.

L'intégralité des actions sont liées par une liste chaînée simple : chaque structure d'information d'une action possédant un pointeur sur la suivante (*voir figure 13*).

Chaque action est conservée dans son propre fichier spécifique, dont le nom se compose de la référence de l'action (son symbole) suivie d'une extension STO. Pour une application commerciale, on utiliserait plus certainement une bibliothèque d'accès à une base de données compatible Windows.

Figure 13 : Structure d'information d'une action.

```
typedef struct tagInCoreStockInfo
{
  char    szFileName[MAXFILENAME];
            /* fichier contenant les
données de l'action */
  STOCKFILE StockFile;
            /* copie de l'en-tête du
fichier d'action */
  HANDLE hTicks;
  HWND   hWnd;
            /* fenêtre d'affichage de
l'action */
  DWORD  dwFlags;
            /* toutes sortes de bits
d'état nécessaires */
#define STATE_HAS_VGRID  1L
#define STATE_HAS_HGRID  2L
  HANDLE hNextStockInfo;
            /* pointeur sur la prochaine
structure */
} STOCKINFO, FAR *LPSTOCKINFO;
```

Utiliser les boîtes de dialogue

Pour mieux comprendre les boîtes de dialogue, nous allons observer l'une figurant dans notre application de portefeuille d'actions. En sélectionnant la rubrique Fichier/Ouvrir, une boîte de dialogue apparaît; elle fournit la liste des fichiers STO trouvés dans le répertoire courant. L'utilisateur doit pouvoir modifier les noms des répertoires et des unités de disque afin de rechercher d'autres fichiers STO. La quasi-totalité des applications Windows possèdent une boîte de dialogue Fichier/Ouvrir. Malheureusement, Windows ne fournit pas de boîte standard et chaque vendeur implémente la sienne, plus ou moins caractéristique. Notre exemple d'application utilise une portion de code typique proposée par le SDK de Windows.

Il faut d'abord définir la boîte de dialogue dans le fichier ressources. La définition de l'une de ces dernières est présentée dans la figure 14.

La boîte de dialogue s'intitule «Ouvrir»; elle possède une barre de titre et un menu système (*voir figure 15, photo page suivante*). Un champ d'édition permet à l'utilisateur de taper un nom de fichier ou de spécifier une autre extension (le texte par défaut est «*.STO»). Ce champ récupère également la sélection courante de la boîte liste, qui contient les noms de tous les fichiers du répertoire courant correspondant à la spécification donnée.

Cette boîte renferme également la liste des sous-répertoires du répertoire courant, ainsi que l'inventaire de toutes les unités de disque du système. Enfin, la boîte de dialogue comporte un bouton poussoir OK permettant d'ouvrir le fichier sélectionné et un bouton Annuler qui arrête l'opération Ouvrir. Le premier constitue le bouton par défaut : si l'utilisateur presse la touche Entrée (ou Enter), un message WM_COMMAND, avec IDOK dans wParam, est envoyé à la procédure utilisateur de la boîte de dialogue.

La Winproc de la fenêtre principale de l'application traite le message WM_COMMAND et invoque la boîte de dialogue Fichier/Ouvrir si wParam a pour valeur ID_OPEN.

```
  case ID_OPEN :
  lpfn = MakeProcInstance((FARPROC) OpenDlg,
  hThisInstance); DialogBox(hThisInstance,
"Ouvrir", hWnd, lpfn);
  FreeProcInstance(lpfn);
  break;
```

On souhaite que la procédure de la boîte de dialogue retourne un handle sur le fichier ouvert plutôt qu'une valeur booléenne True/False ou qu'un identificateur de contrôle. Un handle de fichier renvoyé par la fonction Windows OpenFile constitue une valeur sur un mot (WORD), et il peut donc être fourni à l'application en le passant comme second argument de EndDialog.

Voici l'en-tête de la procédure de la boîte de dialogue :

```
BOOL FAR PASCAL OpenDlg(hDlg, message, wPa-
ram, lParam)
    HWND hDlg;
    unsigned message;
    WORD wParam;
    LONG lParam;
{
    WORD index;
PSTR  pTptr;  HANDLE   hFile;

    switch  (message)
{
    .
    .
    .
```

Les deux seuls messages à traiter par la procédure de la boîte de dialogue sont WM_INITDIALOG et WM_COMMAND. Examinons d'abord la petite fonction UpDateListBox. Elle construit un chemin complet — terminé par un nom de fichier — en effectuant la concaténation de deux variables chaînes, l'une contenant le chemin courant et l'autre le nom complet du fichier. Elle appelle ensuite une fonction Windows intitulée «DlgDirList». DlgDirList reçoit cinq arguments : 1) un handle de boîte de dialogue, 2) l'identificateur d'une boîte liste (celle des des fichiers) dans cette boîte de dialogue, 3) l'identificateur d'un champ de texte statique, également dans cette dernière, 4) une chaîne comprenant un chemin et un nom de fichier et 5) une valeur représentant des attributs de fichier. La fonction scrute ensuite le chemin pour y détecter tous les fichiers dont le nom correspond à la spécification et dont les attributs sont identiques à ceux désirés. Le nom de chaque fichier trouvé se place dans la boîte liste, tandis que le nom du chemin s'affiche dans le champ statique. DlgDirList dispose automatiquement les sous-répertoires et les noms des unités de disques dans la boîte liste.

En dernier lieu, UpdateListBox appelle SetDlgItemText pour mettre la spécification par défaut dans le champ d'édition de la boîte de dialogue. La fonction SetDlgItemText récupère l'identificateur d'un contrôle de la boîte de dialogue, détermine le handle de fenêtre de ce contrôle et lui envoie un message WM_SETTEXT :

FIGURE 14 : Définition de la boîte de dialogue "Ouvrir".

```
Open DIALOG 10, 10, 148, 112
STYLE DS_MODALFRAME | WS_CAPTION |
WS_SYSMENU
CAPTION "Ouvrir "
BEGIN
LTEXT "&Nom du
fichier:",IDC_FILENAME,4,4,60,10
EDITTEXT IDC_EDIT,4,16,100,12,
ES_AUTOHSCROLL
LTEXT "&Fichiers dans",IDC_FILES,4,40,32,10
LISTBOX,IDC_LISTBOX,4,52,70,56,WS_TABSTOP
LTEXT "",IDC_PATH,40,40,100,10
DEFPUSHBUTTON "&Ouvrir",IDOK,87,60,50,14
PUSHBUTTON "Annuler",IDCANCEL,87,80,50,14
END
```

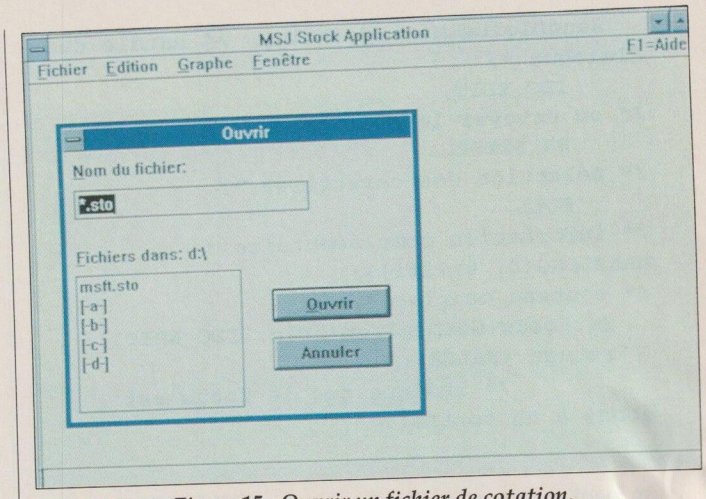


Figure 15 : Ouvrir un fichier de cotation.

```
void UpdateListBox(hDlg)
    HWND hDlg;
{
    strcpy(str, DefPath);
    strcat(str, DefSpec);
    DlgDirList(hDlg, str, IDC_LISTBOX,
IDC_PATH,0x4010);
    SetDlgItemText(hDlg, IDC_EDIT, DefSpec);
}
```

Le message WM_INITDIALOG est alors traité. Un appel à UpdateListBox remplit la boîte avec les noms des fichiers STO et place la chaîne «*.STO» dans le champ d'édition. Le message EM_SETSEL passe ensuite au contrôle d'édition pour mettre tout le champ en surbrillance. Si vous désirez envoyer un message à un contrôle quelconque d'une boîte de dialogue, en ne connaissant que l'identificateur du contrôle, vous pouvez employer SendDlgItemMessage. Cela équivaut à l'instruction suivante :

```
SendMessage(GetDlgItem(hDlg, ID_CONTROL,
msg,wParam, lParam);
```

Vous devez encore attribuer le focus au contrôle d'édition :

```
case WM_INITDIALOG;/* message:
initialisation */
    UpdateListBox(hDlg);
```

```
   SendDlgItemMessage(hDlg,  /* handle du
dialogue */
     IDC_EDIT,
/* où envoyer le message */
     EM_SETSEL,
/* sélection des caractères */
     NULL,
/* information complémentaire */
MAKELONG(0, 0x7fff));
/* contenu complet */
   SetFocus(GetDlgItem(hDlg, IDC_EDIT));
   return (FALSE);
          /* indique que le focus est
donné à un contrôle */
```

On peut à présent traiter le message WM_COMMAND. On souhaite que l'élément retenu de la boîte liste soit affiché simultanément dans le champ d'édition. Quand la sélection change dans une boîte liste, celle-ci le notifie à la boîte de dialogue en lui envoyant un message WM_COMMAND contenant son identificateur dans wParam et le code de notification LBN_SELCHANGE dans le mot de poids fort de lParam (*voir figure 16*). Un moyen simple d'obtenir le texte de la rubrique choisie consiste à appeler la fonction Windows prédéfinie DlgDirSelect.

Pourquoi les messages LB_GETCURSEL et LB_GETTEXT ne sont-ils pas envoyés à la boîte liste? Mis à part le fait qu'il faudrait augmenter la taille du programme, la fonction DlgDirSelect supprime les crochets qui entourent le nom

Figure 16 : Traitement du message WM_COMMAND.

```
case WM_COMMAND:
  switch (wParam)
  {
    case IDC_LISTBOX:
       switch (HIWORD(lParam))
       {
         case LBN_SELCHANGE:
           if (!DlgDirSelect(hDlg, str,
IDC_LISTBOX))
           {
             SetDlgItemText(hDlg,
IDC_EDIT, str);
             SendDlgItemMessage(hDlg,
IDC_EDIT, EM_SETSEL,NULL, MAKELONG(0,
0x7fff));
           }
           else
           {
             strcat(str, DefSpec);
             DlgDirList(hDlg, str,
IDC_LISTBOX, IDC_PATH, 0x4010);
           }
           break;

           case LBN_DBLCLK:
           goto openfile;
       }
       return TRUE;
```

Figure 17 : Ouvrir un fichier en pressant le bouton "OK".

```
                .
                .
                .
        case IDOK:
openfile:
        GetDlgItemText(hDlg, IDC_EDIT,
OpenName, 128);
        if (strchr(OpenName, '*') ||
strchr(OpenName, '?'))
        {
        SeparateFile(hDlg, (LPSTR) str,
        (LPSTR) DefSpec, (LPSTR)
OpenName);
        if (str[0])
        strcpy(DefPath, str);
        ChangeDefExt(DefExt, DefSpec);
        UpdateListBox(hDlg);
        return TRUE;
        }
        if (!OpenName[0])
        {
        MessageBox(hDlg, "Aucun nom de
fichier sp_cifié.",
        NULL, MB_OK | MB_ICONHAND);
        return TRUE;
           }

        AddExt(OpenName, DefExt);

/* Insérer ici la routine d'ouverture du
fichier. */

/* Elle doit retourner le handle au lieu
de NULL. */

        StockFileRead((LPSTR) OpenName);

        EndDialog(hDlg, hFile);
        return (TRUE);

        case IDCANCEL:
            EndDialog(hDlg, NULL);
            return (TRUE);
        }
        break;

    }
    return FALSE;
}
```

d'un répertoire ainsi que les crochets et tirets qui encadrent les noms des unités de disque dans la boîte liste. Elle retourne la valeur zéro si l'élément sélectionné est un nom de fichier et une valeur différente s'il s'agit d'un nom de répertoire. Si la sélection courante représente un nom de fichier, celui-ci se place dans le contrôle d'édition; pour un nom de répertoire, la boîte liste est actualisée.

Un double-clic sur la sélection d'une boîte liste entraîne le même résultat qu'en cliquant une fois sur la sélection, puis sur le bouton OK. Lorsque le code de notification LBN_DBLCLK est reçu, le programme effectue simplement un saut vers la routine chargée d'ouvrir le fichier (voir figure *17*).

En dernier lieu, il faut gérer le comportement des deux boutons poussoirs de la boîte de dialogue. N'oubliez pas que, lorsqu'un bouton est pressé, la fenêtre mère reçoit un message WM-COMMAND, avec la valeur de l'identificateur de contrôle du bouton dans le paramètre wParam. Si l'utilisateur clique sur Annuler, il n'y a qu'à terminer le traitement de la boîte et retourner NULL comme valeur de handle de fichier. Cliquer sur OK, conduit à un travail plus conséquent. Il faut d'abord examiner le contenu du champ d'édition et déterminer si le nom de fichier possède un caractère générique. Si c'est le cas, nous supposons que l'utilisateur souhaite rechercher un nouveau nom de fichier. Le programme divise alors la spécification de fichier en un chemin et un nom de fichier, puis remplit la boîte liste avec les fichiers correspondant à cette dernière spécification. Si le champ d'édition est vide, il faut présenter une boîte de messages pour avertir l'utilisateur et rendre la main au gestionnaire de boîte de dialogue. Dans les autres cas le champ d'édition contient un nom de fichier valide : le programme ouvre le fichier et retourne son handle.

Et voilà, nous disposons de notre première boîte de dialogue. Avec un peu d'habitude, le code de programmation des boîtes de dialogue s'écrit presque machinalement.

Ajouter une cotation

A la fin d'une journée d'opérations boursières, il faut entrer dans la base de données la valeur de clôture et le volume des échanges de chaque action. Pour cela, sélectionnez la rubrique Ajouter une cotation du menu Edition. La boîte de dialogue Ajouter une cotation est affichée et contient les informations concernant l'action courante (*voir figure 18, photo ci-contre*); vous pouvez remplir les différents champs afin d'ajouter une nouvelle cotation en fin de liste. La boîte de dialogue Ajouter une cotation possède trois champs : la date d'échange, le volume et le prix final (pour l'instant, ce dernier doit être une valeur entière). A ce stade intermédiaire de l'application, le programme ne prend pas en compte les champs de date et de volume — seul le prix final a de l'importance. Pour le moment, une cotation ne peut pas être insérée au milieu de la liste des cotations. Nous y remédierons dans une future version du programme.

La boîte de dialogue Options

La boîte de dialogue Options permet de contrôler les caractéristiques d'une action et le type d'affichage de son graphe (*voir figure 19, photo ci-contre*). En fait, la même boîte est présentée lorsqu'on ajoute une nouvelle action à la base de données ou quand on change les propriétés d'une action existante. La seule différence réside, pour le second cas, dans la désactivation du champ contenant le nom de l'action : cette information ne peut pas être modifiée pour une action existante. Les lignes de code suivantes désactivent le contrôle symbole (la référence) lorsque nous changeons les caractéristiques d'une action :

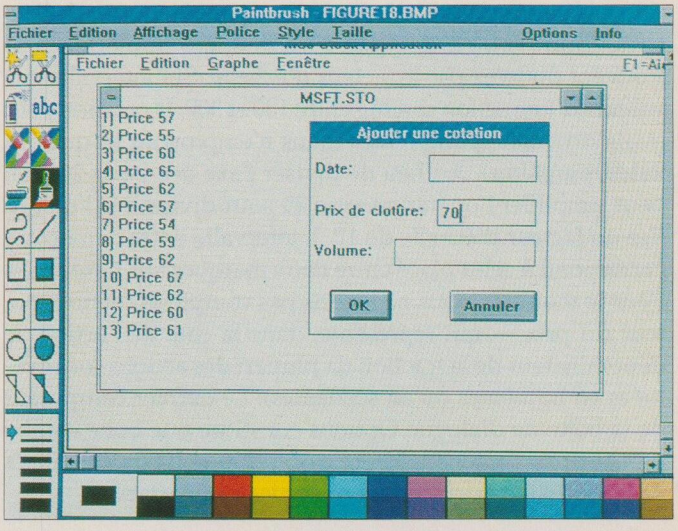


Figure 18 . La boîte de dialogue "Ajouter une cotation".

```
if (lpStockInfo -hTicks != NULL)
    .
    .
    .
 EnableWindow(GetDlgItem(hDlg, ID_SYMBOL),
FALSE);
```

Comme il a été précisé plus haut, certains champs de la boîte de dialogue sont ignorés à ce stade de développement du programme. Les valeurs minimales et maximales déter-

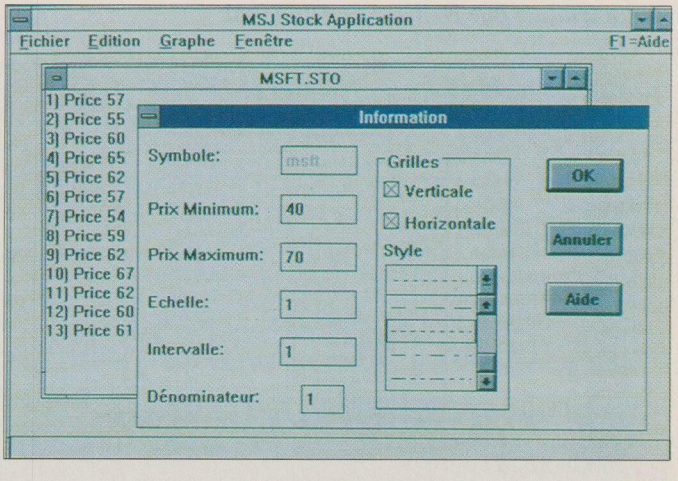


Figure 19. La boîte de dialogue Options permet de contrôler les caractéristiques d'une action et le type d'affichage de son graphe.

minent l'intervalle utile de l'axe des y pour le graphe de l'action. Avant l'affichage, toutes les données numériques se trouvent divisées par le facteur d'échelle. Si l'intervalle des valeurs d'une action se situe entre 100 et 300, la cotation peut avoir 201 valeurs différentes (nous n'employons ici que des valeurs entières). Au lieu de diviser l'axe des y par 201, on peut simplifier l'opération avec 20 pour diviseur si l'on utilise un facteur d'échelle de 10. L'intervalle entre deux cotes correspond à celui placé entre deux marques sur l'axe des y. Pour le moment, nous ne tenons pas compte du dénominateur du prix — qui représente, dans la cote de l'action, le dénominateur de la fraction (la plupart des actions sont évaluées en huitièmes ou en seizièmes). Le dernier composant de la boîte de dialogue Options constitue une boîte combo owner-draw contenant les styles des différents stylos utilisés pour les grilles horizontales et verticales du graphe.

Tester l'application

La fonction StockFileRead lit un fichier d'action et le charge dans l'application. StockFileRead commence par allouer de la mémoire pour l'en-tête de l'action, puis consulte ce dernier. Le champ numérique de l'en-tête est examiné afin de s'assurer qu'il s'agit d'un fichier d'action valide. Le programme réserve de la mémoire pour les informations de cotation avant de lire ces dernières. Ensuite, il appelle la fonction GraphCreateWindow afin de créer une fenêtre fille MDI associée à une action. Le handle de cette fenêtre est placé dans la structure d'information de l'action. De la même manière, le handle mémoire de cette structure se trouve stocké dans la zone des octets supplémentaires à l'aide de la fonction SetWindowWord. Ainsi, il deviendra possible de déterminer rapidement quelles informations sont associées à une fenêtre particulière. Une fenêtre d'action affichée présente un texte concernant l'action.

Texte traduit par Henry-Luc Planchat

L'un des fichiers nécessaire à la création de STOCK.EXE

```
/*      Fichier     : FILEOPEN.C              */
#include <string.h>
#include <windows.h>
#include "stock.h"
char FileName[128];
char PathName[128];
char OpenName[128];
char DefPath[128];
char DefSpec[13] = "*.sto";
char DefExt[] = ".sto";
char str[255];

HANDLE FAR PASCAL OpenDlg(hDlg, message,
 wParam, lParam)
  HWND hDlg;
  unsigned message;
  WORD wParam;
  LONG lParam;
{
  WORD index;
  PSTR pTptr;
  HANDLE hFile;
  switch (message)
  {
    case WM_COMMAND:
      switch (wParam)
      {
        case IDC_LISTBOX:
          switch (HIWORD(lParam))
          {
            case LBN_SELCHANGE:
              if (!DlgDirSelect(hDlg, str,
                    IDC_LISTBOX))
              {
                SetDlgItemText(hDlg,
                     IDC_EDIT, str);
                SendDlgItemMessage(hDlg,
                 IDC_EDIT,EM_SETSEL,NULL,
                 MAKELONG(0, 0x7fff));
              }
              else
              {
                strcat(str, DefSpec);
                DlgDirList(hDlg, str,
                     IDC_LISTBOX, IDC_PATH,
                     0x4010);
              }
              break;

            case LBN_DBLCLK:
              goto openfile;
          }
          return TRUE;

        case IDOK:
openfile:
          GetDlgItemText(hDlg, IDC_EDIT,
              OpenName, 128);
          if (strchr(OpenName, '*') ||
              strchr(OpenName, '?'))
          {
            SeparateFile(hDlg, (LPSTR)
              str, (LPSTR) DefSpec, (LPSTR)
              OpenName);
            if (str[0])
              strcpy(DefPath, str);
            ChangeDefExt(DefExt, DefSpec);
            UpdateListBox(hDlg);
            return TRUE;
          }
          if (!OpenName[0])
          {
            MessageBox(hDlg, "Aucun nom de
              fichier spécifié.", NULL,
              MB_OK | MB_ICONHAND);
            return TRUE;
          }
           AddExt(OpenName, DefExt);
/* Insérer ici la routine d'ouverture du
fichier. Elle doit retourner le handle. */
          StockFileRead((LPSTR) OpenName);
          EndDialog(hDlg, hFile);
          return (TRUE);

        case IDCANCEL:
          EndDialog(hDlg, NULL);
          return (TRUE);
        }
        break;
```

```
        case WM_INITDIALOG:
          UpdateListBox(hDlg);
          SetDlgItemText(hDlg, IDC_EDIT,
            DefSpec);
          SendDlgItemMessage(hDlg,IDC_EDIT,
            EM_SETSEL,NULL,
            MAKELONG(0, 0x7fff));
          SetFocus(GetDlgItem(hDlg,
            IDC_EDIT));
            return (FALSE);
      }
    return FALSE;
}
void UpdateListBox(hDlg)
  HWND hDlg;
{
  strcpy(str, DefPath);
  strcat(str, DefSpec);
  DlgDirList(hDlg, str, IDC_LISTBOX,
       IDC_PATH, 0x4010);
  SetDlgItemText(hDlg, IDC_EDIT, DefSpec);
}
void ChangeDefExt(Ext, Name)
  PSTR Ext, Name;
{
  PSTR pTptr;
  pTptr = Name;
  while (*pTptr && *pTptr != '.')
    pTptr++;
  if (*pTptr)
    if (!strchr(pTptr, '*') &&
        !strchr(pTptr, '?'))
      strcpy(Ext, pTptr);
}
void SeparateFile(hDlg, lpDestPath,
 lpDestFileName, lpSrcFileName)
  HWND hDlg;
  LPSTR lpDestPath, lpDestFileName,
        lpSrcFileName;
{
  LPSTR lpTmp;
  char  cTmp;
  lpTmp = lpSrcFileName +
          (long) lstrlen(lpSrcFileName);
  while (*lpTmp != ':' && *lpTmp !=
       '\\' && lpTmp > lpSrcFileName)
    lpTmp = AnsiPrev(lpSrcFileName, lpTmp);
  if (*lpTmp != ':' && *lpTmp != '\\')
  {
    lstrcpy(lpDestFileName, lpSrcFileName);
    lpDestPath[0] = 0;
    return;
  }
  lstrcpy(lpDestFileName, lpTmp + 1);
  cTmp = *(lpTmp + 1);
  lstrcpy(lpDestPath, lpSrcFileName);
  *(lpTmp + 1) = cTmp;
  lpDestPath[(lpTmp - lpSrcFileName)+1]= 0;
}
void AddExt(Name, Ext)
  PSTR Name, Ext;
{
  PSTR pTptr;
  pTptr = Name;
  while (*pTptr && *pTptr != '.')
    pTptr++;
  if (*pTptr != '.')
    strcat(Name, Ext);
}
```


dBGEN 6.1

Générateur d'applications

dBASE III+, DBASE IV, FoxPro, NANTUCKET

Générez vos applications dBASE sans programmer grâce à l'interface de paramétrage **QuickGEN**

- **Multi-fichiers** (en relation)
- **Multi-postes** (gestion des conflits)
- **Multi-écrans** (fixes ou déroulants)
- **Pop-menus** (1+2 niveaux d'appel)
- **Fenêtrage/sélection** (illimité)

➠ Simplicité, rapidité et souplesse

Les fichiers et index étant décrits dans un dictionnaire, il vous suffit pour définir un programme d'indiquer les fichiers en relation et la présentation formulaire ou déroulante pour chacun d'eux. Un éditeur WYSIWYG vous permet de tracer les écrans et positionner les champs en fenêtrage sur fichiers. Appels de fonctions et programmation dBase sont possibles.

Il ne vous reste plus qu'à choisir votre langage et générer les .PRG. Le code généré est simple, modifiable et exécutable sans "run time".

➠ Un résultat sans compromis

Enfin la minute de vérité est arrivée ! Vous découvrez une magnifique application avec menus déroulants, fenêtrage, pagination, gestion transparente du réseau ... du professionnel !

➠ Un outil sans limitation

Vous ferez vos maquettes en un temps record bien sûr, mais aussi vous irez jusqu'au bout de vos développements grâce à l'interface d'accès direct au dictionnaire. Ici vous entrez dans l'univers dBGEN, plus aucune limitation d'écriture, vous disposez de toute la puissance de l'atelier logiciel, (nombreux points de programmation, utilisation des IF, DO, WHILE, DO CASE, nombre illimité d'écrans, etc...). Avec documentation automatique

➠ Et la souris ...

En standard dans dBGEN, en option dans vos applications .

Marques déposées : NANTUCKET Corp, dBFAST (BUMBLEBEE), dBASE III+ & dBASE IV (ASHTON TATE), FoxBASE (FoxSOFTWARE).

Présentation les deuxièmes vendredi de chaque mois de 10 H à 16 H
HOTEL FRANTOUR 163 bis, avenue Clichy PARIS 17
Réservation : ALTI-SOFT 78 72 05 69

Je désire recevoir : une documentation ☐ une version de prise en main ☐
Ci joint un chèque de 150 F pour une prise en main

Sté _____
Adresse _____
_____
Nom _____

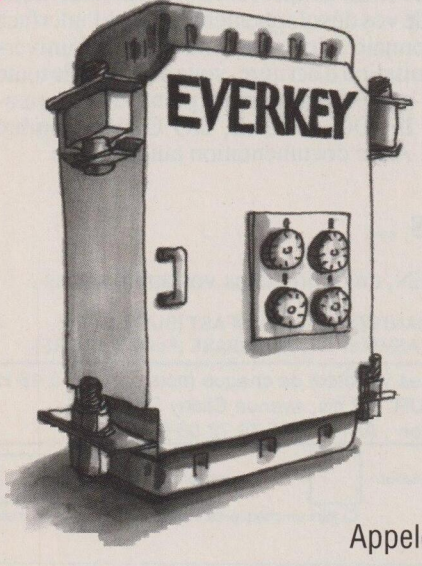
ALTI-SOFT 32, av. Félix FAURE 69007 LYON
Tél. :78 72 05 69

Service lecteurs n° 11 (dernière page)

INFO PC
LE MAGAZINE DE LA MICRO-INFORMATIQUE PROFESSIONNELLE
LANGAGES ET SYSTÈMES
Immeuble La Fayette, 2, place des Vosges, Cedex 65, 92051 Paris La Défense 5,
Tél. : 49.04.79.00 Télex : 613 234 F MONDINF
DIRECTEUR : JACQUES TILLIER assisté de CATHERINE BOISADAM,
RÉDACTEUR EN CHEF : PHILIPPE MONIN, **ASSISTANT DE LA RÉDACTION :** SERGE DEL'HOMMEAU ,
1ère SECRÉTAIRE DE RÉDACTION : CATHERINE OLZCHANECH
SECRÉTAIRES DE RÉDACTION : ELISABETH DÉROFF, ISABELLE DUMONTEIL, STÉPHANIE GRIN
ASSISTANT DE LA FABRICATION : STÉPHANE GAUTIER
CE NUMÉRO A ÉTÉ RÉALISÉ SOUS LA RESPONSABILITÉ DE YVES GUITTARD
Ont également collaboré à ce numéro : Marc Adler, François Bergeon,
Alain Dorseuil, Luc Lemaire, Henry-Luc Planchat, Claude Simon, Matthieu Thfoin, Edouard Tonneau
L'article traitant de l'initiation à Windows 3 correspond à la traduction d'articles
provenant de Microsoft Systems Journal, une publication de Microsoft Corp.
La rédaction d'INFOPC dégage sa responsabilité en ce qui concerne les documents écrits, les matériels
et les logiciels qui lui seraient envoyés sans demande préalable de sa part.
PUBLICITÉ : DIRECTEUR DE LA PUBLICITÉ : SOPHIE TALAYRACH
DIRECTEUR ADJOINT DE LA PUBLICITÉ : NATHALIE TESSLER
RESPONSABLE COMMERCIAL LANGAGES & SYSTÈMES : HERVÉ LABATTUT, **CHEFS DE PUBLICITÉ** :VIRGINIE BRIL,
THIEN GILBERT, **ASSISTANTES DE PUBLICITÉ :** AROMI RAJON, LAURENCE CARTON
DIFFUSION : DIRECTEUR : JEAN PAITEL, **RESPONSABLE SERVICE ABONNEMENTS** :
LUCIENNE BOSSER assisté de JOSIANE LIBAN, ROSE MOLNAR, ALAIN PACCAGNELLA,
BERNADETTE THUAL **RESPONSABLE DES VENTES** : PIERRE-OLIVIER PROVENT
Tél. : 49.04.79.68, Abonnements : 49.04.79.67, 49.04.79.72, Suisse :
MADITEC SA - Case postale 144 CH1020 RENENS 1 - Tél. : (021)35.08.65, Canada et Etats-Unis : Editions Gosselin Inc. 228, rue de la Colline,
AYLMER PQ j9j 1t8, CANADA - Tél. : 819-684-7885
INFOPC est une publication éditée par Micropresse S.A. Capital de 250000 F, en location-gérance auprès de
IDG
ISSN 0981 - 6402
Commission paritaire n° 67338

NOUVELLE TECHNOLOGIE

EVERKEY : LE BLINDAGE DE VOS LOGICIELS



GRACE A LA NOUVELLE TECHNOLOGIE "ZERO-LOAD/SINGLE WIRE", EVERKEY DEVIENT LA CLE DE PROTECTION IDEALE DE TOUS VOS LOGICIELS DOS ET WINDOWS.

- **Protection** maximale triple avec encryption.
- **Comptabilité** avec tous les PCs, et portables.
- **Sérialisation** unique au laser de chaque clé.
- **Programmation** et paramétrage à distance.
- **Transparence** parfaite en application réseau et mono.

KIT D'EVALUATION COMPLET
logiciel et manuel gratuits
(587,07 FTTC) 495 FHT

Appelez ou écrivez dés maintenant à : **INNOSOFT**

Tél. (1) 40 99 28 00 - 2, rue des Bourets - 92150 Suresnes - Fax : 40 99 28 88

BON DE COMMANDE - A renvoyer à INNOSOFT 2, rue des Bourets - 92150 Suresnes

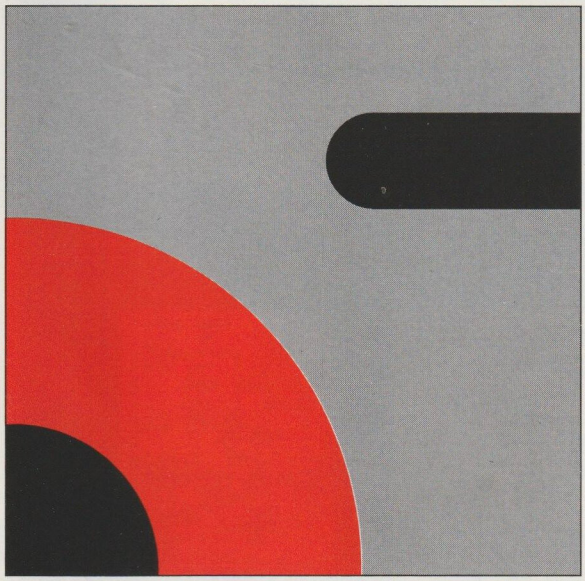
Je commande I__I kit d'evaluation complet EVERKEY au prix unitaire de : 652,30 FHT (587,07 F TTC + 65,23 F TTC de frais de port)
Règlement : I__I Chèque I__I Contre-remboursement (60 F TTC de frais de supplémentaires) I__I Carte Bleu N° : .....Date d'expiration : .....
Nom : ..... Adresse : .....
Société : ..... Code postal : I_I_I_I_I_I Ville : .....

INFORMATICIENS

POUR VOTRE CARRIERE

Rencontrez en Direct

COMPU SEARCH



LE SALON DE L'EMPLOI INFORMATIQUE

DES ENTREPRISES QUI RECRUTENT

Les Responsables du Recrutement et les Dirigeants des entreprises du secteur informatique, SSII, Constructeurs et des Entreprises Utilisatrices vous recevront "en direct" pour répondre à vos attentes en termes de métier, de carrière, de formation et d'évolution.

DES ENTREPRISES QUI CONSEILLENT

Les Consultants des plus grands cabinets experts de la gestion des carrières, de l'évaluation des potentiels, des échelles de salaires et du recrutement dans le secteur informatique seront à votre disposition pour vous conseiller gratuitement et sans rendez-vous pendant toute la durée de **COMPU SEARCH.**

10 et 11 Janvier 1992

VENDREDI DE 10 H A 20 H - SAMEDI DE 10 H A 18 H
PALAIS DES CONGRES - PORTE MAILLOT - PARIS 17ème
ENTREE LIBRE ET GRATUITE.

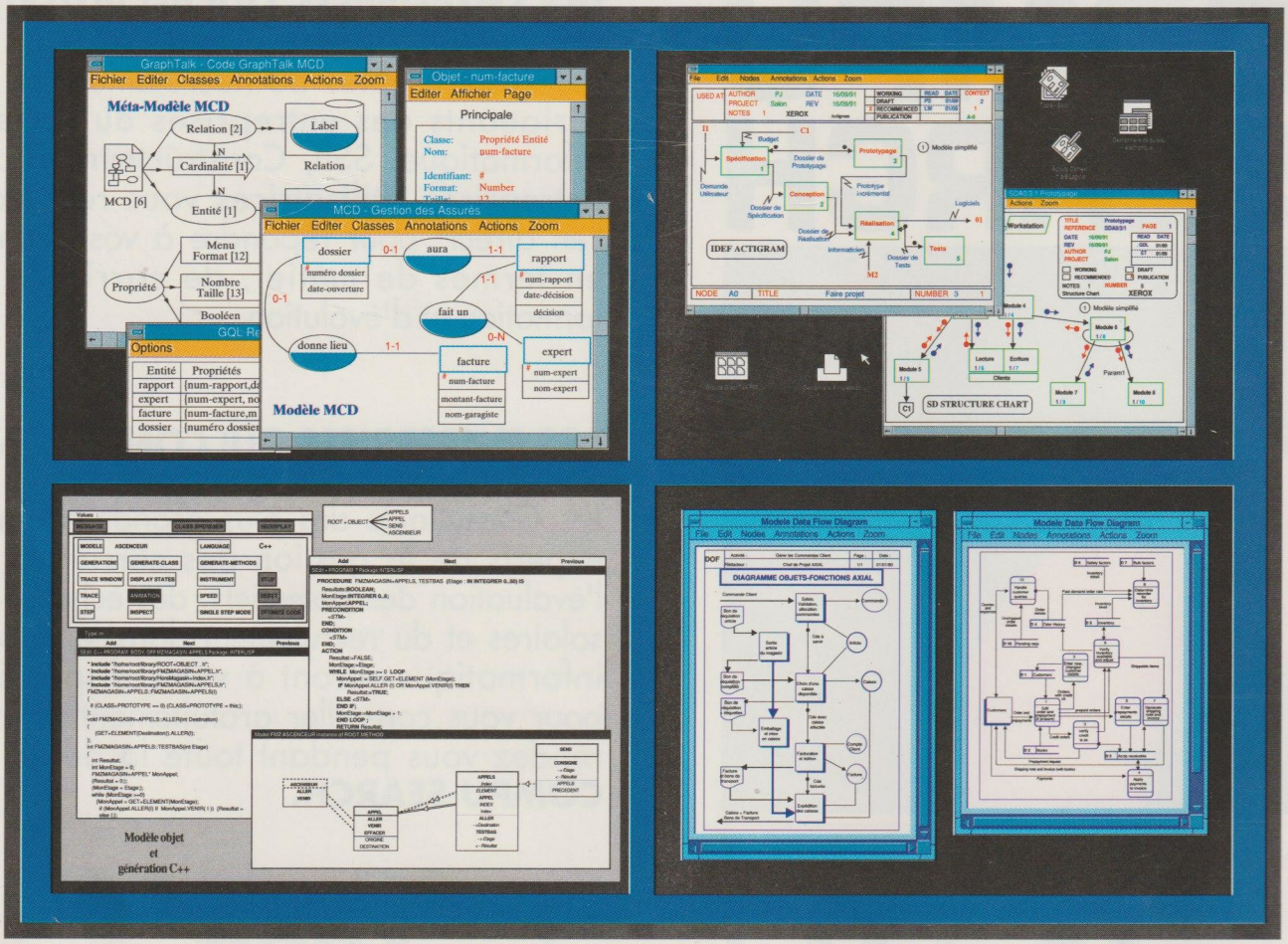
INFORMATIONS : JL. HUGODOT - Tél. : 49 04 79 07 - IDG CARRIERES - Cedex 65 - 92051 PARIS LA DEFENSE.

Service lecteurs nº 13 (dernière page)

RANK XEROX

La technologie GraphTalk

Générateur d'ateliers
Générateur de codes
Générateur de documents
Catalogue d'A.G.L.



Ateliers GraphTalk:
MERISE, AXIAL, NIAM, SA, SSADM, IDEF, SD, ROOT, OSSAD.

Plates-formes supportées:
PC, PS, stations de travail supportant OS2/PM, UNIX/MOTIF, MS/Windows.

Compatibilité:
Tout développement réalisé sur un système est immédiatement opérationnel sur les autres sans conversion.

Rank Xerox DIA/GL 7, rue Touzet 93586 Saint-Ouen Cedex Tél.: (1) 49 48 50 85